



Verification and computation in restricted Tile Automata

David Caballero¹ · Timothy Gomez¹ · Robert Schweller¹ · Tim Wylie¹

Accepted: 30 September 2021
© The Author(s), under exclusive licence to Springer Nature B.V. 2021

Abstract

Many models of self-assembly have been shown to be capable of performing computation. Tile Automata was recently introduced combining features of both Cellular 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.

Keywords Tile automata · Turing machines · Unique assembly verification

1 Introduction

Self-assembly systems have quickly become an intense area of research due to fabrication simplicity (Kanaras et al. 2003), the ability to create systems at the DNA level (Kimna and Lieleg 2019), the control of nanobots (Kawano 2018), and the maturity of experimental techniques (Evans 2014). 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) (Winfree 1998), the 2-Handed Assembly Model (2HAM) (Cannon et al. 2013), the Staged self-assembly model (Demaine et al. 2011), and the Signal-passing Tile Assembly Model (STAM) (Padilla et al. 2013). 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* (Chalk et al. 2018a), 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 example of this type of application includes (Alumbaugh et al. 2019) in which TA is shown capable of simulating the *Amoebots* model (Daymude et al. 2019) of

✉ Timothy Gomez
timothy.gomez01@utrgv.edu

David Caballero
david.caballero01@utrgv.edu

Robert Schweller
robert.schweller@utrgv.edu

Tim Wylie
timothy.wylie@utrgv.edu

¹ Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, USA

programmable matter. More recently a connection between the STAM and Tile Automata was established in (Cantu et al. 2020) where it was shown that the STAM is capable of simulating any Tile Automata system.

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.

This is the full extended version of the conference paper (Caballero et al. 2020) with additional results, proofs, and content to make the paper more accessible. Additional results and proofs may be found in Sects. 4 and 8.

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 (Winfree 1998), and the computational power is explored in depth in other works such as Keenan et al. (2016). The 2-Handed Assembly Model (2HAM) (Cannon et al. 2013) introduced a more powerful model and is capable of fuel efficient computation (Schweller and Sherman 2013) along with the Signal-passing Tile Assembly Model (Padilla et al. 2013), which has tiles that can interact to turn glues on or off.

In Demaine et al. (2011), Winslow (2015), 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, with only a constant number of tile types, a system can construct length- n lines using $\mathcal{O}(\log n)$ bins and mixes (Demaine et al. 2008). Repulsive forces have been shown to aid in constructing shapes at constant scale (Luchsinger et al. 2018). Further, by utilizing the temperature to encode information, shapes can be constructed with constant (or nearly) tile types (Chalk et al. 2018b; Schweller et al. 2019a).

The Unique Assembly Verification (UAV) problem asks if a given system uniquely produces a given assembly. In the aTAM, this problem is solvable in polynomial time (Adleman et al. 2002). In the 2HAM, however, UAV was shown to be in coNP with certain generalizations being coNP-complete (Cannon et al. 2013; Schweller et al. 2017). In the staged assembly model, this problem is known to be PSPACE-complete (Caballero et al. 2021;

Schweller et al. 2019b). Adding the power of negative glues also vastly changes the complexity of this problem making it uncomputable in models that include it due to the ability for pieces of assemblies to break off (Doty et al. 2013). 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 (Cantu et al. 2020).

Cellular Automata has been shown to be Turing Universal even in one dimension (Cook 2004). Other stronger results have been shown proving intrinsic universality of many versions of Cellular Automata (Goles et al. 2011; Worsch 2013).

The Tile Automata model was introduced in Chalk et al. (2018a) 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 (Alumbaugh et al. 2019). A model motivated by real-world implementations, the Signal-passing Tile Assembly Model, is able to simulate Tile Automata (Cantu et al. 2020) meaning results shown in the TA model carry over to STAM at scale.

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 1-dimensional 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 the largest number of symbols before halting for a given number of states). In the more restricted case of freezing systems, we design a Turing machine simulation that utilizes only height-2 assemblies. We also present freezing systems that can construct n -length lines using $\mathcal{O}(n)$ states. Results are shown in Table 1.

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 1 and 9, respectively

Turing machine	Tile Automata system	States	Transition rules
Deterministic	Non-Freezing 1D	$\mathcal{O}(Q \Gamma)$	$\mathcal{O}(\delta)$
Deterministic	Freezing height-2	$\mathcal{O}(Q \Gamma)$	$\mathcal{O}(\delta)$

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**

Transition rules	Freezing	1D Result	2D Result	Theorem
Affinity Strengthening	Freezing	P_{\parallel}^{NP} -Complete	coNP ^{NP} -Complete	Theorems 11 and 10
Affinity Strengthening	Non-freezing	PSPACE-Complete	PSPACE-Complete	Theorem 8
General	Freezing	Open	Undecidable	Theorem 7
General	Non-freezing	Undecidable	Undecidable	Theorem 6

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 (even with a max height of 2 for all assemblies). 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 a bounded-space Turing machine simulation. When restricting the model to both affinity strengthening and freezing, we show membership in coNP^{NP}. We then provide a reduction to show coNP^{NP}-completeness even for height-2 Tile Automata systems. When restricted to one dimension, we show the UAV problem is in the class P_{\parallel}^{NP} and provide a reduction from Max True 3SAT Equality (Spakowski 2006) to show P_{\parallel}^{NP} -completeness. P_{\parallel}^{NP} is the class of problems solvable by a polynomial time Turing machine with parallel access to an NP oracle, that is it may make a single oracle call to a polynomial number of NP oracles that run in parallel.

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-

indicates whether the system is freezing where tiles cannot repeat states. **Result 1D** is the complexity of UAV in one dimension and **Result 2D** is the complexity of UAV in two dimensions. **Theorem** is where these results can be found

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 Fig. 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.

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).

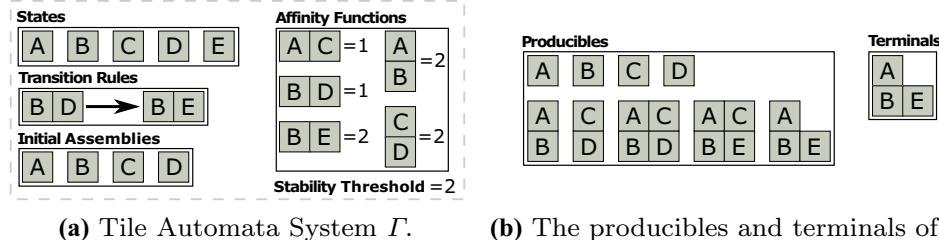


Fig. 1 An example of a tile automata system Γ . Recursively applying the transition rules and affinity functions to the initial assemblies of a system yields a set of producible assemblies. Any producibles that

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 $\text{SHAPE}_{\mathcal{A}}$. 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 $x_1 > x_2$.

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 $\mathbf{v} = (v_1, v_2)$, let $\mathcal{A}_{\mathbf{v}}$ denote the positioned assembly obtained by translating each tile in \mathcal{A} by vector \mathbf{v} . An *assembly* is a set of all translations $\mathcal{A}_{\mathbf{v}}$ 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 or positioned shape 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_C for some positioned assembly $C \in C$ 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 .

(b) The producibles and terminals of Γ .

cannot combine with, break into, or transition to another assembly are considered to be terminal

Combinable Assemblies. Two assemblies are τ -combinable provided they may attach along a border whose strength sums to at least τ . Formally, two assemblies A and B are τ -combinable into an assembly C provided G_C for any $C \in C$ has a cut $(\mathcal{A}, \mathcal{B})$ 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 Fig. 1.

Definition 1 (Tile Automata Producibility) For a given tile automata system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$, the set of *producible assemblies* of Γ , denoted PROD_{Γ} , is defined recursively:

- (Base) $\Lambda \subseteq \text{PROD}_{\Gamma}$
- (Recursion) Any of the following:
 - (Combinations) For any $A, B \in \text{PROD}_{\Gamma}$ such that A and B are τ -combinable into C , then $C \in \text{PROD}_{\Gamma}$.

- (Breaks) For any $C \in \text{PROD}_\Gamma$ such that C is τ -breakable into A and B , then $A, B \in \text{PROD}_\Gamma$.
- (Transitions) For any $A \in \text{PROD}_\Gamma$ such that A is transitional into B (with respect to Δ), then $B \in \text{PROD}_\Gamma$.

For a system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$, we say $A \rightarrow_1^\Gamma B$ for assemblies A and B if A is τ -combinable with some producible assembly to form B , if A is transitional 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 \rightarrow^Γ to be the transitive closure of \rightarrow_1^Γ , i.e., $A \rightarrow^\Gamma B$ means that A may grow into B through a sequence of combinations, transitions, and/or breaks.

Definition 2 (Production Graph) The production graph of a Tile Automata system Γ is a directed graph where each vertex corresponds to an assembly in PROD_Γ and there exists a directed edge between assemblies A and B if $A \rightarrow^\Gamma B$.

Definition 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 transitional to any producible assembly of Γ . Let $\text{TERM}_\Gamma \subseteq \text{PROD}_\Gamma$ denote the set of producible assemblies of Γ that are terminal.

Definition 4 (Transition Stable Assembly) An assembly A is Transition Stable if it is not transitional to any producible assembly of Γ .

Definition 5 (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 that it held previously. This implies that any given tile in the system can only undergo a finite number of state transitions.

Definition 6 (Affinity Strengthening) An Affinity Strengthening system is a Tile Automata system where all transition rules must maintain or increase a state's 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' , $\Pi(s, t) \leq \Pi(s', t) \forall t \in \Sigma$.

Definition 7 (Bounded) A tile automata system Γ is *bounded* if and only if there exists a $k \in \mathbb{Z}_{>0}$ such that for all $A \in \text{PROD}_\Gamma$, $|A| < k$.

Definition 8 (Height- k Systems) A tile automata system is said to have Height- k if all producible assemblies have a height of less than or equal to k .

Definition 9 (Unique Assembly) A Tile Automata system Γ uniquely produces an assembly A if

- A is the only assembly in TERM_Γ
- for all $B \in \text{PROD}_\Gamma$, $B \rightarrow^\Gamma A$.
- Γ is bounded.
- there does not exist a pair of assemblies $B, C \in \text{PROD}_\Gamma$, such that $B \rightarrow^\Gamma C \rightarrow^\Gamma B$.²

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 Padilla et al. (2013); Schweller and Sherman (2013) for fuel-efficiency, and Cantu et al. (2020) for covert computation.

Definition 10 (Simulation) A Tile Automata system T is said to simulate a Turing machine M if every producible assembly a of T can be mapped to a configuration m of M and for 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 11 (Covert Computation) Given a Tile Automata system T that simulates a Turing machine M , T covertly simulates M if there exists two assemblies A and R , such that for any input x that M accepts, A is the unique terminal assembly when M is simulated on x , and for any input y that M rejects R is the unique terminal assembly.

Definition 12 (Fuel Efficient Computation) A fuel efficient Turing machine simulation in Tile Automata

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

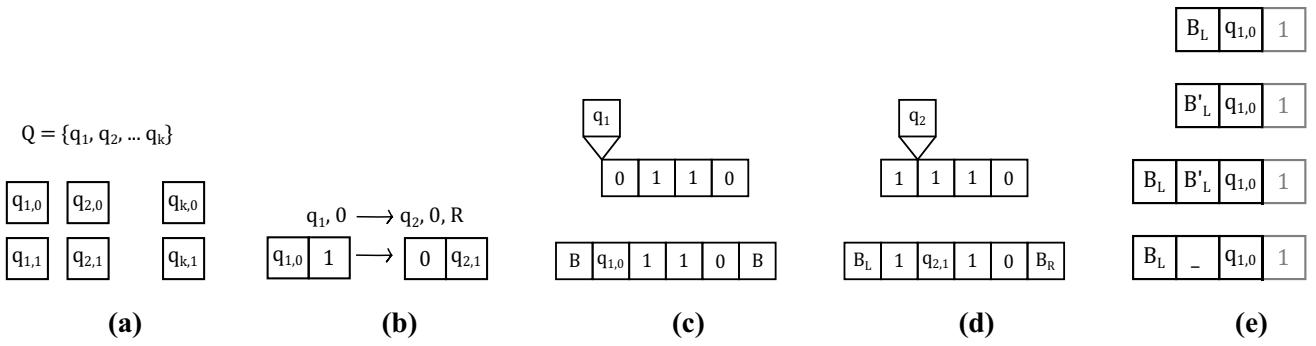


Fig. 2 **a** Tile automata states (below) created from the states of Turing machine (above) over a binary alphabet. **b** State change rules (below) created from the Turing machine transition rules (above). **c** A Turing machine (above) configuration and the representative TA

assembly (below). **d** The same Turing machine (above) after making one step and the assembly (below) after the same step. **e** The process that takes place to extend the tape

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 1 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 $T = (\Sigma_{TA}, \Pi, \Lambda, \Delta)$ that can simulate M such that $|\Sigma_{TA}| = \mathcal{O}(|Q||\Gamma|)$ and $|\Delta| = \mathcal{O}(|\delta|)$.³

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 $\mathcal{S} = \{\sigma_s | s \in \Sigma\}$ (Fig. 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 (Fig. 2b). WLOG, assume an assembly A representing 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 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 (Fig. 2c, d).

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. The process can be seen in Fig. 2e. 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

³ One-dimensional Tile Automata systems always have $\tau = 1$, so we omit that parameter from T .

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 change the state of 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. \square

4 Thin freezing turing machines

In this section we explore the abilities of bounded height freezing Tile Automata systems. We present a freezing Tile Automata system that can simulate general Turing machines that have producible assemblies of height-2.

4.1 Overview

This construction functions in a similar way to the previous Turing machine simulation for non-freezing systems.

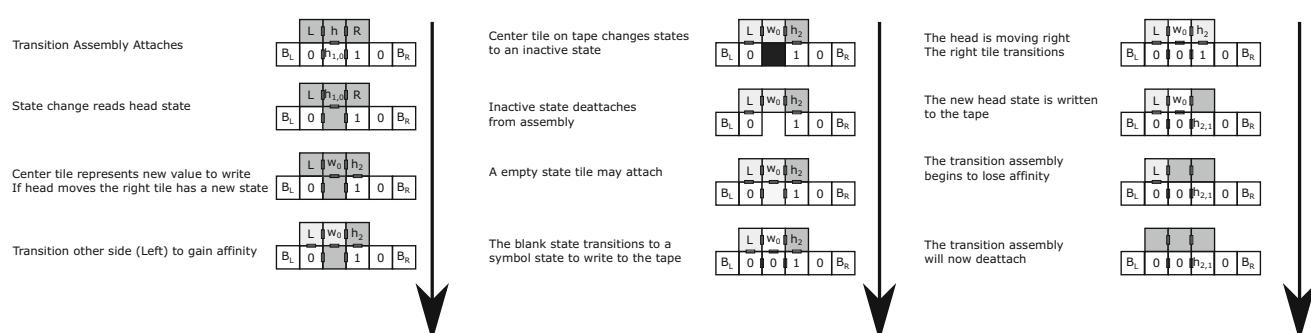


Fig. 3 Transition process for one computation step of the height-2 freezing Turing machine. The transition assembly starts by attaching to the tape assembly through the head state to start the process. Once

However, in this case transitions do not take place between tiles on the tape but through the help of a transition assembly. This transition assembly attaches in the row $y = 1$ while the Turing machine tape exists in row $y = 0$. This system also uses $\tau = 1$. This system can be broken into three parts: the *Tape Assembly*, which functions similar to the previous construction, the *Transition Assembly*, which is used to control the steps of the computation and replace tiles, and the *Inactive States*, which are the removed tiles that can no longer transition. The Transition Assembly attaches to the Tape Assembly to perform a single step of the computation. In doing so tiles are removed from the tape assembly resulting in an inactive state being produced. Since our system is freezing the transition assembly will eventually reach an inactive state as well. At the end of our computation our output states will clean up these inactive states to ensure our system uniquely produces a single assembly.

4.2 Tape assembly

As with the previous construction, we have head states and symbol states that represent the current state of the Turing machine and the contents of the tape. However, here these tiles do not have transition rules with each other and only change states via the transition assembly. The input assembly is constructed in the same way as the previous simulation by encoding the initial tape contents with symbol states and a single head state representing the starting location and state of the Turing machine head. We call this row of tiles the *tape assembly*. Our system further includes empty state tiles. These tiles are different from blank states and are used by the transition assembly to write a new value to the tape during computation.

4.3 Transition assembly

The *transition assembly* is made of three unique tiles and exists as an initial assembly in our system. The center tile

the process is complete, the transition goes to inactive states that fall off the assembly, which exposes a new head state so another transition assembly may attach

has affinity with the head states of our tape. The process of transitioning the assembly can be seen in Fig. 3. This can be seen as three parts, Attachment, Writing, and Movement.

In the first *Attachment* step, the transition assembly attaches to the tape assembly via affinity with the center tile. Once attached, the center tile transitions to a state representing the transition rule of the Turing machine and changes the head state. The center tile then transitions in the direction of the head movement. W.L.O.G. assume the head moves right. The right and center tiles will transition and the right tile will now represent the next state of the head and gains affinity with the tape assembly. The opposite direction (in this case left) tile then transitions to gain affinity with the tape assembly. This transition is the last of the attachment step and the center tile transitions to start the next step.

In the *Writing* step, the previous head state transitions to an inactive head state. This state does not have affinity with symbol states or the states of the transition assembly so it detaches from the assembly. This allows for an empty state tile to attach. This tile changes states with the center state of the transition assembly to write the new value to the tape. The center tile also transitions to another state signaling the start of the next step.

The final *Movement* step starts with the center tile transitioning with the right tile (still assuming the head moves right). The right tile may now transition with the tape assembly to change the tile directly underneath into a head state. This transition also changes the right tile of the transition assembly to an inactive state. This transition propagates through the assembly, which then falls off after losing affinity. The inactive states of the transition assembly have affinity with each other but not with the tape assembly. This exposes the next head state so a new transition assembly can attach, allowing the computation to continue.

If the next head state would result in the Turing machine accepting, an accept state will appear on the tape and the same process occurs as in the non freezing construction to deconstruct the assembly into single accept state tiles.

4.4 Inactive states and clean up

Since this is a freezing system we use detachment to replace tiles that have reached what we call an inactive state. These inactive states do not have affinity with the states of the tape assembly. Each tile of the transition assembly has an inactive state that it transitions to at the end of the computation step. Further, the previous head states of the Turing machine go to an inactive state when replaced. These tiles only attach to accept/reject states that have already fallen off the tape assembly after the

computation is complete. The accept/reject states “clean up” the inactive states so there is only one terminal assembly.

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

Proof The initial assemblies in this system are the initial tape assembly, which encodes the input to M , the transition assembly, and empty state tiles. Each tape assembly maps to a configuration of M . For each transition step of M , a transition assembly will attach to the tape assembly to remove the old head state, attach a new tile in that position, write the new symbol, and finally write the next head state. The transition assembly then detaches from the tape assembly and the new head state on the tape allows for another transition assembly to attach.

If the head of M were to move off the edge of the tape, the process to extend the assembly is similar to the previous construction as well. The system still includes the buffer states B_L, B_R, B'_L and B'_R . When the Transition assembly would write the new head state to the tape if it is above a buffer state B_L or B_R , the buffer state will transition to B'_L or B'_R respectively. This will allow another buffer state to attach then transition the previous state to the state representing the blank symbol. The transition assembly then writes the head state to the tape and begins detachment.

The only possible terminal assemblies for this system are either the accept state with inactive tiles attached to it if M accepts, or the same assembly with a reject state if M rejects. Each possible output of M has a unique assembly so this simulation is covert. This system is fuel efficient as well since at each computation step the only attachments that occur are the transition assembly (of size 3), and the empty tile that attaches to write a new symbol to the tape. \square

5 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 1-dimensional assemblies and leverage Theorems 1 and 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 3 to show that this problem is

uncomputable for general Tile Automata even in one dimension.

5.1 General

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

Definition 13 (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 1 *For any n -state 2-symbol (not including the blank symbol) Turing machine M that produces an output x , there exists a $\mathcal{O}(n)$ -state Tile Automata System T that uniquely assembles an assembly A , such that A represents x .*

Proof We modify the construction from Theorem 1, 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, and is bounded by $\mathcal{O}(n)$. Note there is no need for accept or reject states since the head state turns into a symbol state when the TM halts. \square

Theorem 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 1, we can take any Busy Beaver Machine and create a Tile Automata system that uniquely produces an assembly the same size as the number of symbols printed on the tape. \square

5.2 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. This construction draws some inspiration from the result in Demaine et al. (2008). For clarity, we begin with a helping lemma.

Lemma 2 *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 Fig. 4. The only initial states are 1_A and 1_B . The affinities are between adjacent states. Pairs of tile that have transition rules between them are highlighted, with the resulting assembly shown as well. Our unique terminal assembly is a length 2^3 line. By adding a constant number of states, transitions, and affinities to this system, the length of the uniquely assembled line will double, and this process can be repeated to uniquely assemble any length 2^x line.

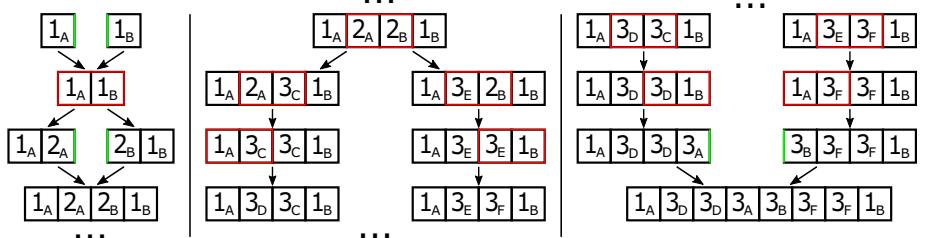
For $n > 3$ and $x = \log n$, let T_n be the system that uniquely assembles a length 2^x line derived by recursively applying the following process to T_3 ($x - 3$) times. Assuming that T_n uniquely assembles a length 2^x line of the form $(1_A, n_D, \dots, n_D, n_A, n_B, n_F, \dots, n_F, 1_B)$, T_{n+1} 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 $n + 1_E$ that allow the state to propagate left and right, respectively, and then transition to $n + 1_D$ and $n + 1_F$, respectively, when the end of 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^x lines to bond and uniquely assemble a length 2^{x+1} line. This new system uniquely produces a length 2^{x+1} line of the same form as previously described, and the process can be repeated to double the length of the unique assembly again. \square

Theorem 4 *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 2 to build arbitrary length- n lines. To build any length- n line using $\mathcal{O}(\log n)$ states, let $T = T_{\lceil \log_2 n \rceil}$. Let b_i indicate the i^{th} least significant bit of n 's binary expansion (the left most bit is b_0). For all $i \geq 1$ 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, they exist in an assembled line of length 2^i . This transition “locks” this producible, and stops it from growing. Four more transition rules are added to allow this state to propagate to the ends of the line. Finally, we add 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 assembly to its left, and the next smallest locked assembly on its right. The locked assembly that is the most significant bit (leftmost), does not attach to anything on its left side. The same for the least significant on

⁴ For this definition we consider Turing machines using a binary alphabet.

Fig. 4 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 states that have transition rules between them are highlighted in red, with the next assembly showing the resulting states



its right. For the case of the bit b_0 , if it is equal to one we will add an additional state, 0_L , which starts in our initial assemblies and acts as a length-1 locked assembly. \square

5.3 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 5 *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 3, there 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. \square

6 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 in TA, even in one dimension, 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.

6.1 Undecidability

Theorem 6 *Tile Automata Unique Assembly Verification is undecidable in one dimension.*

Proof Using Theorem 1, 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 that is the final accept state tile that is our target assembly. If M does not halt, then there exists no terminal assemblies. This is true

under both definitions of uniquely assembly since the only time a cycle exists in the production graph of Γ is if M ever revisits a configuration. If M revisits a configuration, then M will not halt, and thus our system will not uniquely assemble the final accept state tile. \square

Theorem 7 *Freezing 2-dimensional Tile Automata Unique Assembly Verification is undecidable even when all assemblies are of max height-2.*

Proof Using Theorem 9, we can perform a similar reduction as above. Given a Turing machine M , we construct a freezing Tile Automata system Γ that simulates M . If M halts, Γ uniquely produces an assembly with the inactive states attached to the accept state. This assembly is our target assembly. If M does not halt, Γ does not have any terminal assemblies and does not uniquely produce the target assembly. \square

7 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 of the UAV problem. We explore Affinity Strengthening Tile Automata (ASTA). We start by considering the non-freezing case, then consider the added restriction of freezing in the following section.

Lemma 3 *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 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 $2|A|$ need to be checked since any assembly larger than $2|A|$ would have been built using at least one assembly of size greater than $|A|$, which would have already been rejected. We can check if B is terminal by nondeterministically 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, $B \xrightarrow{\Gamma} A$. By continuing to perform construction steps on B , we will eventually reach A . If we ever reach B again, 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. \square

Lemma 4 *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 $L \in \text{PSPACE}$. Consider a Turing machine M that decides L in polynomial space. The construction from Theorem 1 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 Kuroda (1964)). 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 results in two

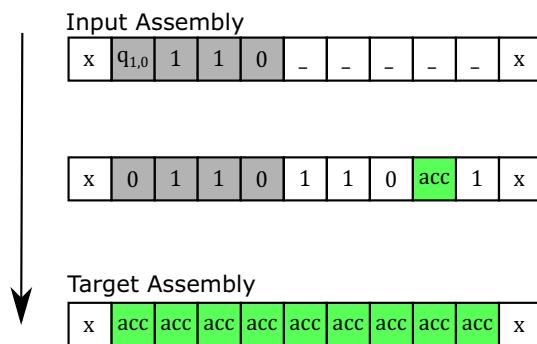


Fig. 5 An overview of the construction in Lemma 4. The reduction starts with a fixed-length tape. Once the machine accepts/halts, the accept state appears on the assembly. This causes a transition that “hides” the tape by transitioning to the target assembly shown

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. The assembly with the accept states will be our target assembly. 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 . Figure 5 shows our input assembly, the assembly where the accept state first appears, and the target assembly.

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 exists 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. \square

Theorem 8 *The Unique Assembly Verification problem in Affinity Strengthening Tile Automata is PSPACE-complete.*

Proof Follows from Lemmas 3 and 4. \square

8 Freezing ASTA UAV

In this section, we explore the Unique Assembly Verification problem for freezing affinity strengthening systems. We begin by presenting a bounded time Turing machine simulation and use that to design a *SAT evaluator*, a freezing ASTA system that provides the information for which assignments satisfy a given formula. This SAT evaluator system is used as a basis for the two reductions in this section. We first show coNP^{NP}-completeness for the UAV problem in systems with a max assembly height of 3, utilizing the SAT evaluator to show hardness. We then explore this problem in one dimension. We show membership in the class P_{||}^{NP}, which is the class of problems solvable by a deterministic Turing machine in polynomial time using a polynomial number of NP oracles that can only be accessed a single time in parallel. We then provide a reduction from a known P_{||}^{NP}-complete problem.

8.1 Freezing ASTA UAV membership

We first show that the freezing ASTA UAV problem is in the class coNP^{NP}.

Lemma 5 *The Unique Assembly Verification problem in freezing Affinity Strengthening Tile Automata is in coNP_{NP}.*

Proof We use the algorithm from Lemma 3 to prove that the running time is polynomial for freezing systems. When building an assembly B , since the system is freezing, 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 a subroutine 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 the subroutines as oracles, this problem is in coNP_{NP}. \square

8.2 Freezing ASTA SAT evaluator

The next two hardness results will utilize a method for creating a 1D freezing affinity-strengthening system Γ that evaluates the assignments that satisfy a formula ϕ . This system will produce a terminal assembly for each assignment to ϕ that contains a flag for whether that assignment evaluates to true or false. We call these *assignment assemblies*. We first show how 1D freezing ASTA can simulate bounded-time Turing machines.

Theorem 9 *For any bounded-time Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_a, q_r, q_s)$, there exists a 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 $|\Delta| = \mathcal{O}(|\delta|TIME(M)^2)$.*

Proof We modify the construction from Theorem 1. 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. Assume we are given an oblivious Turing machine M , which means the movement of the head is a function of the current step of the Turing machine. Thus, we know when certain cells will be written to and can expand the state space of the Tile Automata system so that a tile will never need to repeat states. Since we know the run time of M is bounded, we only need a finite number of states for each cell. The states on the tape will keep track of the current symbol on the tape, if the current head state is at that tile, and how many times the cell has been modified. If the head of M moves to cell c a total of x times, then the tile representing c will have $x|\Sigma|$ symbol states and $x|\Gamma||\Sigma|$ head states.

This increase in state-space ensures no tile will ever become the same state twice since each cell will only need to change states a finite number of times. \square

8.2.1 SAT evaluator construction

Consider an oblivious Turing machine M that solves the Circuit Value problem for a circuit with n variables that computes ϕ , and then prints T or F to the tape representing this evaluation.

We want to simulate M on the 2^n different inputs that are length- n bit strings (assignments to ϕ). By Theorem 9, there is a 1D freezing Tile Automata System that can simulate this Turing machine with the starting input assembly representing a single input to ϕ . Now consider a modification to the system with 2^n input assemblies in Γ representing every possible input to ϕ . This will now result in a system that computes all 2^n assignments, but contains an exponential number of starting assemblies. This can be fixed, however, using the nondeterminism inherent to Tile Automata.

This process is shown in Fig. 6. We start with $2n$ dominoes, an initial 0 and 1 domino for each variable. Starting with the two dominoes representing the first variable x_1 , one of them will be selected for attachment. Then either of the two tiles representing x_2 may attach to the first domino. This process repeats to construct a length- n line. Since at each position one of the two tiles will be selected, this results in a total of 2^n possible assemblies. Using this method, we can create all 2^n possible input assemblies, which can then be used as input assemblies to a simulation of M . This method is only used to construct the section of the input assembly containing the input to M . The other tiles, such as the buffer states and the work tape of M , attach to this section to form the full input assembly (Fig. 6).

At the end of the computation simulation, we have either a True (T tile) or False (F tile) state on each assembly. Then the T and F state propagate left and right through transition rules, erasing computation history, but preserving the section of the assembly that contains the original assignment. The assembly is flagged true or false depending on the presence of the T or F , respectively.

Using this method, we black box this functionality for evaluating a SAT formula with 2^n assemblies (representing each possible assignment), which are flagged true or false depending on whether that assignment evaluates to true or false. We say the system $\Gamma(\phi)$ is the system created from formula ϕ .

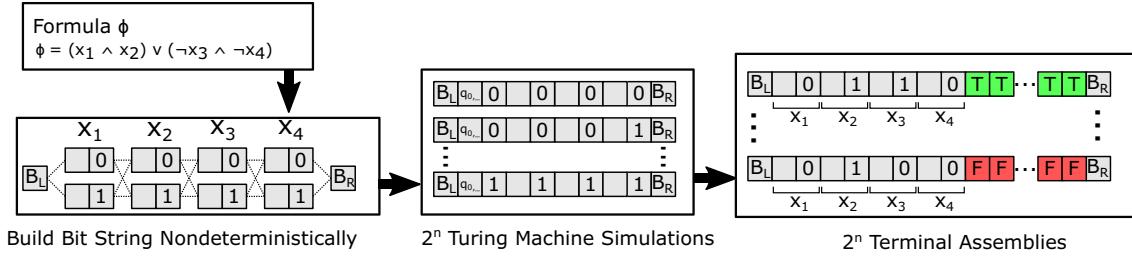


Fig. 6 An overview of the steps taken by the SAT Evaluator. First an assembly for each possible input is built, then the Turing machine is simulated on all inputs. If any of the assemblies result in a terminal accept assembly, then the instance of SAT is true

8.3 Height-3 freezing ASTA UAV is coNP^{NP}-complete

In this section we prove that a special case of freezing ASTA UAV, where the system is guaranteed to only produce assemblies of height ≤ 3 , is coNP^{NP}-complete. Lemma 5 shows coNP^{NP} membership. We now present a reduction from $\forall \exists \text{SAT}$ to show hardness.

Definition 14 ($\forall \exists \text{SAT}$) Given a 3SAT formula $\phi(x_1, \dots, x_k, x_{k+1}, \dots, x_n)$, is it true that for every assignment to variables x_1, \dots, x_k , there exists an assignment to x_{k+1}, \dots, x_n such that $\phi(x_1, \dots, x_n)$ is satisfied?

8.3.1 Overview

We utilize the SAT evaluator construction to derive a system that has a terminal assembly for every assignment flagged as true or false. We modify the system so that those assemblies grow geometry that encodes their assignment in geometric bumps and dents. A second set of assemblies, called test assemblies, is also constructed. A test assembly encodes an assignment to x_1, \dots, x_k . It can attach to an assembly representing an assignment if it is flagged true, and has the same assignment to x_1, \dots, x_k .

8.3.2 Creation of height-3 freezing ASTA system

Given an instance of $\forall \exists \text{SAT} = \forall x_1, \dots, x_k \exists x_{k+1}, \dots, x_n (\phi)$, create the system $\Gamma(\phi)$ as defined in Sect. 8.2. Recall the current terminal assemblies of the system are the 2^n assignment assemblies— each flagging their corresponding assignment as True/False, and then they can grow the geometry on each assignment assembly that reflects the assignment (Fig. 7a). We add transition rules with the output states (T/F), which propagate left and transition the states that represent assignments to the variables in x_1, \dots, x_k , and allowing single tiles to attach from above. This is done from left to right, with a transition happening after each attachment that is required for the process to continue. This acts as a confirmation that each single tile is attaching. The tiles encode the assignment geometrically like so: The tile will attach directly above the variable if its value is one, and will attach above the tile on its left if the value is zero. The pattern of tile positions encodes a binary string. Once all tiles have attached, transitions will propagate to the leftmost output tile (T/F), changing state T to state A , and state F to state B .

Test Assemblies. We add additional initial tiles to the system that grow into test assemblies (Fig. 7b). These test assemblies have a section for each variable (a horizontal domino). The sections representing a variable in x_1, \dots, x_k have an additional tile attached that represents an

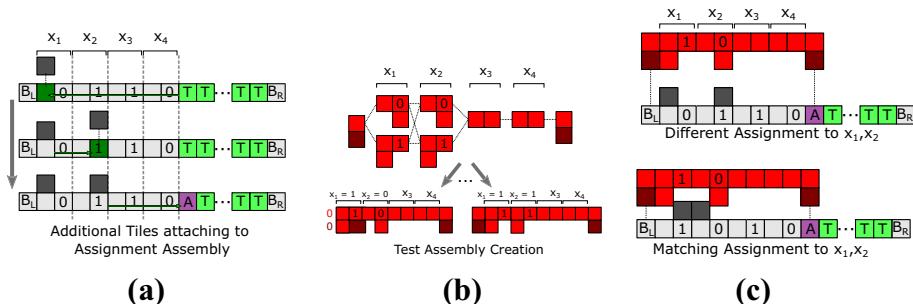


Fig. 7 Example of $\forall \exists \text{SAT}$ over 4 variables with $k = 2$. (a) Transition rules pass a signal to the left that transitions the tile next to the state representing x_1 , allowing a tile to attach. This allows more transitions that allow a tile to attach over the state representing x_2 (since it is $= 1$). Finally, the leftmost T is transitioned to state A . (b) Four test assemblies are built, each representing an assignment to x_1, x_2 . A test assembly can attach to an assignment assembly with the A state if they encode the same assignment to x_1, x_2

$= 1$. Finally, the leftmost T is transitioned to state A . (b) Four test assemblies are built, each representing an assignment to x_1, x_2 . A test assembly can attach to an assignment assembly with the A state if they encode the same assignment to x_1, x_2

assignment. The positioning is complementary to that of the assignment assemblies (1 = tile on the left, 0 = tile on the right). They grow two tiles on each side, and the lower tiles on the left/right have affinity 1 with the B_L/A state, respectively.

The system will also build one “blank” test assembly that has no tiles encoding the assignment for any of its variables. It can connect using affinity with either the A state or the B state, so it attaches to every assignment assembly. This means the assignment assemblies that represent a false output are not terminal.

Test Assembly/Assignment Assembly Interaction The assignment assemblies, or test assemblies, can attach to each other utilizing affinity involving the B_L state and the A state, which appears only after all additional tiles have attached. Figure 7c shows how attachment is not possible if the partial assignment to x_1, \dots, x_k is not the same—the tiles representing unequal variable assignments would overlap, and therefore the two assemblies are not able to attach. If the partial assignment matches, the two assemblies fit together.

Transition to Target Assembly The target assembly is the size of an assignment assembly flagged true with a test assembly attached. Each tile is enumerated; if B_L is the point of origin, then each tile at position x, y is the state $X_{x,y}$ in the target assembly (Fig. 8). An assignment assembly flagged True attaches to a test assembly, and a transition with the state B_L occurs, transitioning the two states to $X_{0,0}$ and $X_{0,1}$. These states propagate throughout the entire assembly. For each state $X_{x,y}$, it has transition rules with any state it is possible for its neighbors to be in, and transitions them to $X_{x\pm 1,y}$ or $X_{x,y\pm 1}$, based on its relative position. The “blank” test assemblies attach to any assignment assembly, but the resulting assembly will be missing tiles relative to the target assembly, so there is a single “filler” tile in the set of initial assemblies. State $X_{x,y}$ has affinity with the filler tile on any side where that location is nonempty in the target assembly. Therefore, the

filler tile occupies the empty spots and allows the assembly to transition to the target.

Theorem 10 *Unique Assembly Verification in height-3 freezing Affinity Strengthening Tile Automata is coNP^{NP}-complete.*

Proof This reduction takes as input an instance of $\forall \exists \text{SAT}$ $P = \forall x_1, \dots, x_k \exists x_{k+1}, \dots, x_n (\phi)$, and outputs an instance of height-3 freezing Affinity Strengthening Tile Automata UAV $P' = (\Gamma, A)$, such that the instance of P is true \iff the instance of P' is true.

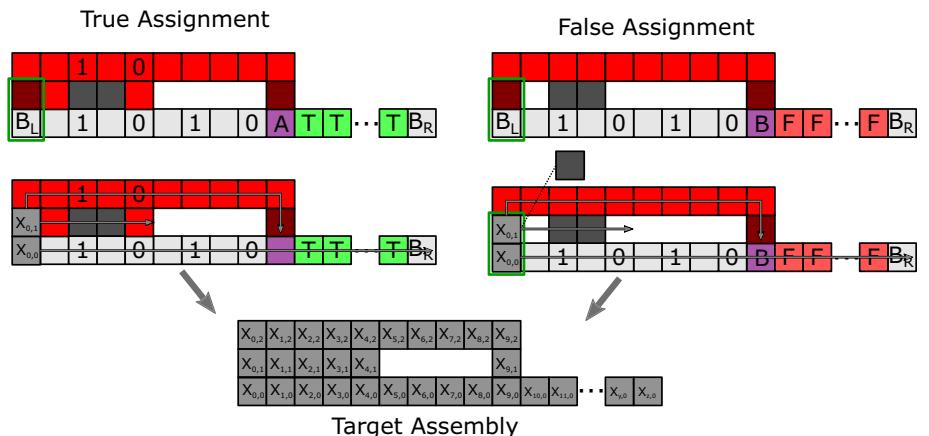
Note that all assemblies representing an assignment to ϕ will grow to the target, as the “blank” test assembly can attach to every one of them and allow it to grow to the target. The test assemblies that represent an assignment to x_1, \dots, x_k only grow to the target assembly if they can attach to some assignment assembly that evaluated to true.

If instance P is true, then for all assignments x_1, \dots, x_k , there exists some assignment to x_{k+1}, \dots, x_n that evaluated to true. Therefore, for all test assemblies, there exists some assignment assembly it can attach to, and all test assemblies grow to the target assembly. The target assembly is uniquely assembled, so P' is true.

If instance P is false, there exists an assignment x_1, \dots, x_k with no satisfying assignment to variables x_{k+1}, \dots, x_n . The test assembly that represents assignment x_1, \dots, x_k is not able to attach to any assignment assembly including any assignment assembly with a different assignment to x_1, \dots, x_k due to geometric blocking. Since all assignments that start with x_1, \dots, x_k evaluate to false, there is no assignment with the A state that the test assembly can attach to. The test assembly representing the assignment x_1, \dots, x_k is terminal, and is not the target assembly, so the instance P' is false.

This shows that Unique Assembly Verification in height-3 freezing Affinity Strengthening Tile Automata is coNP^{NP}-hard, while Lemma 5 shows membership for

Fig. 8 An overview of how all assignment assemblies transition to the target assembly. Assignment assemblies marked true can grow to the target assembly by attaching to a test assembly. Those flagged false can attach to “blank” test assemblies, and the filler tile fills in the spots missing relative to the target assembly



coNP^{NP} . Therefore the problem is coNP^{NP} -complete.

8.4 One-dimensional freezing ASTA UAV

In this section we will show 1-dimensional freezing ASTA UAV is complete for P_{\parallel}^{NP} . We provide a P_{\parallel}^{NP} algorithm, as well as reduce from a known P_{\parallel}^{NP} -complete problem Max True 3SAT Equality.

Definition 15 $(P \underset{\parallel}{\underset{\parallel}{NP}})$ Class of problems solvable by a deterministic Turing machine in polynomial time that is allowed a single query to a polynomial number of parallel NP oracles.

8.4.1 Hardness

Definition 16 (Max True 3SAT Equality) For a 3-CNF formula F , let $\text{max-1}(F)$ denote the maximum number of variables set to true in a satisfying assignment to F . Given two 3-CNF formulas F_1 and F_2 , is it true that $\text{max-1}(F_1) = \text{max-1}(F_2)$?

Max True 3SAT Equality is complete for the class P_{\parallel}^{NP} [Spakowski \(2006\)](#). Given an instance of this problem, the reduction provides an instance of 1D freezing ASTA UAV.

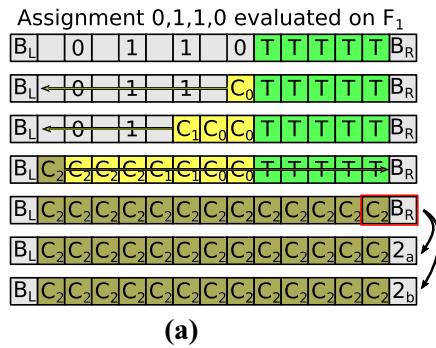
8.4.2 Overview

At a high level, given an instance of Max True 3SAT Equality (F_1, F_2), we utilize the SAT evaluator construction in Sect. 8.2 to represent each assignment/formula pair with an assembly, and flag these assemblies as True or False. We modify the system to allow it to count the number of ones that are contained in the true assemblies, and add a “count” state to the left/right edge of these assemblies that reflects this number. To reach the target assembly, an assembly representing an F_1 assignment has to find a counterpart assembly representing an F_2 assignment to attach to. This attachment is done through the count state that represents the number of ones in the assignment. The affinities between separate count states are designed such that all assemblies can find a necessary counterpart if and only if $\max\mathbb{I}(F_1) = \max\mathbb{I}(F_2)$.

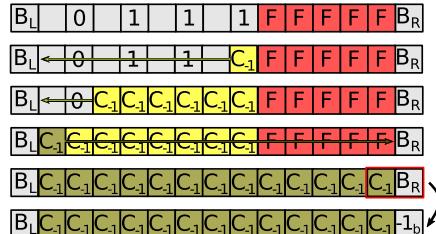
8.4.3 Creation of 1D Freezing ASTA System

From F_1 and F_2 over n variables, we create two SAT evaluator systems $\Gamma_1 = \Gamma_S(F_1)$ and $\Gamma_2 = \Gamma_S(F_2)$, as defined in Sect. 8.2. We merge these two systems into Γ in a way that allows them both to operate independently within the same system. This is done by distinguishing the states of Γ_1 and Γ_2 , and letting Γ be the union of Γ_1 and Γ_2 's states, transition rules, affinity rules, and initial assemblies. Due to this, states that we reference with the same label, e.g. the T/F states, are different states in Γ_1 and Γ_2 . For simplicity, however, we will reference them by this common label.

Fig. 9 (a) An assembly that represents a true assignment to F_1 counts the number of 1's on the tape and sets the rightmost state accordingly. (b) An assembly that represents a true assignment to F_2 counts the number of 1's on the tape and sets the leftmost state accordingly. (c) An assembly that represents a false assignment to F_1 ignores the number of 1's and sets the rightmost state to -1_b . (d) An assembly that represents a false assignment to F_2 ignores the number of 1's and sets the leftmost state to -1_b



Assignment 0,1,1,1 evaluated on F₁



Assignment 0,1,1,0 evaluated on F_2										
B_L	0	1	1	0	T	T	T	T	B_R	
B_L	0	1	1	C_0	T	T	T	T	B_R	
B_L	0	1	C_1	C_0	T	T	T	T	B_R	
B_L	C_2	C_2	C_2	C_2	C_1	C_1	C_0	T	T	B_R
2_a	C_2	C_2	C_2	C_2	C_1	C_1	C_0	T	T	B_R

Assignment 0,1,1,1 evaluated on F_2									
B_L	0	1	1	1	F	F	F	F	B_R
B_L	0	1	1	C₁	F	F	F	F	B_R
B_L	0	C₁	C₁	C₁	C₁	C₁	F	F	B_R
B_L	C₁	C₁	C₁	C₁	C₁	C₁	F	F	B_R
$_{-1}b$	C₁	C₁	C₁	C₁	C₁	C₁	F	F	B_R

At this point, the system will produce 2×2^n unique terminal assemblies. Let $A_{y,x}$ be the assembly that represents the assignment $x \in \{0,1\}^n$ to F_y . For all $x \in \{0,1\}^n$, $(F_y(x) \iff A_{y,x} \text{ contains state } T \text{ and } \neg F_y(x) \iff A_{y,x} \text{ contains state } F)$. We refer to an assembly that contains state T/F as a true/false assembly, respectively.

True assemblies are now flagged with the number of ones they contain. We modify Γ for this purpose. Transition rules are added involving the states that are contained in terminal assemblies (this means they are not terminal with the included changes). For the assemblies $A_{1,x}$, if the “T” state exists, transition rules are added that allow a signal to be passed down the assembly. This signal counts the number of 1s the assignment x contains (let N be that number), and then propagates right. When it reaches the right end of the assembly, it nondeterministically transitions the right buffer state B_R to one of two “count” states, “ N_a ” or “ N_b ” (Fig. 9a). If instead, the “F” state exists, the assembly passes a signal to the right and changes the right buffer state B_R to the count state “ -1_b ” (Fig. 9c).

Assemblies $A_{2,x}$ act in a similar, but complementary, way. The difference is that the transition to count states (“ N_a ”, “ N_b ”, and “ -1_b ”) happen with the left buffer state B_L rather than the right buffer state B_R (Fig. 9b, d).

To the affinity function $\Pi \in \Gamma$, for all values $N, M \in \{-1, 0, 1, \dots, n\}$, and their corresponding count states N_a, M_a, N_b, M_b , we add the following affinity rules conditionally (Note that state -1_a does not exist, so the affinity rules that would include that state are not added):

- $\Pi(N_a, M_a, \vdash) \iff N \geq M$
- $\Pi(N_a, M_b, \vdash) \iff N > M$
- $\Pi(N_b, M_a, \vdash) \iff N < M$
- $\Pi(N_b, M_b, \vdash) \iff N \leq M$

Additional transition rules are added to Γ such that when two count states are adjacent, they transition to state X . State X transitions with nearly every other state, transitioning said state to X . This is done to wipe any computation history from the assembly. The only states it does not

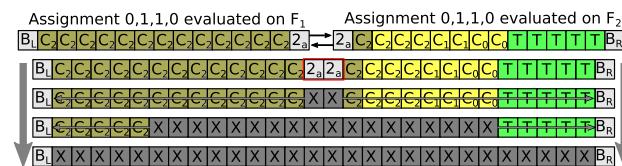


Fig. 10 Two assemblies connecting to reach the target assembly. The top right assembly is produced because $F_1(0, 1, 1, 0) = \text{True}$, and the count state 2_a represents the number of ones in the assignment. This can attach to the assembly on the right, which is produced because $F_2(0, 1, 1, 0) = \text{True}$. Since that assignment also has two 1's, the corresponding count state is on its left edge. The affinity rules described allow for these two to attach

transition with are the buffer states B_L and B_R . This allows the attached $A_{1,x}$ and $A_{2,x'}$ assembly to build towards the target assembly for the created instance of UAV, a 1-dimensional assembly of the form $B_L, X, X, \dots, X, X, B_R$ (Fig. 10).

Theorem 11 *Unique Assembly Verification in 1-dimensional freezing Affinity Strengthening Tile Automata is $P_{\parallel}^{\text{NP}}$ -hard.*

Proof The construction shown takes as input an instance of Max True 3SAT Equality P_{MAX} and provides an instance of 1D freezing Affinity Strengthening UAV $P_{UAV} = (\Gamma, A_T)$, where Γ is the Tile Automata system described, and A_T is the target assembly. Let $N = \max-1(F_1)$ (-1 if F_1 is unsatisfiable) and $M = \max-1(F_2)$ (-1 similarly). For a boolean assignment $x \in \{0, 1\}^n$, and for a boolean string $x \in \{0, 1\}^*$, we denote the number of 1s in x as $|x|_1$.

Let $A_{y,x}$ be the assembly that evaluated $F_y(x)$. Let R_θ be a transition stable assembly with count state θ on its right edge, and L_θ be a transition stable assembly with count state θ on its left edge. Note that all $A_{1,x}$ and $A_{2,x}$ assemblies grow to some R_θ and $L_{\theta'}$, respectively. A R_θ assembly can attach to a $L_{\theta'}$, if θ and θ' meet the conditions for the affinity rule $\Pi(\theta, \theta', \perp)$ to be added. When they attach, their count states are adjacent, allowing both states to transition to state X , which further allows the assembly to grow to the target assembly A_T . Therefore, for every producible R_θ and $L_{\theta'}$, ensuring they can each attach to some counterpart, ensures that the target assembly is uniquely assembled.

If the instance P_{MAX} is true, then $N = M$. Therefore, there exists some producible $R_{N_a}, R_{N_b}, L_{M_a}$, and L_{M_b} .

- For all producible $R_{N'_b}$ assemblies, $R_{N'_b}$ attaches to some L_{M_b} . Either $N' = M = -1$ (both F_1 and F_2 are unsatisfiable), or $N' \leq M$ (since M is the max). Therefore, for all producible $R_{N'_b}$ assemblies, $R_{N'_b}$ is not terminal.
- For all producible $R_{N'_a}$ assemblies: if $N' = M$ then $R_{N'_a}$ attaches to some L_{M_a} . If $N' < M$, there exists some producible $L_{M'_b}$ such that $N' > M'$ (Either $\exists x(\neg F_2(x))$ or $\exists x(|x|_1 < N' \wedge F_2(x))$ must be true).
- For all producible $L_{M'_b}$ assemblies, $L_{M'_b}$ attaches to some R_{N_b} , Either $M' = N = -1$ (both F_1 and F_2 are unsatisfiable), or $M' \leq N$ (since M is the max).
- For all producible $L_{M'_a}$ assemblies: if $M' = N$ then $L_{M'_a}$ attaches to some R_{N_a} . If $M' < N$, there exists some producible $R_{N'_b}$ such that $M' > N'$ (Either $\exists x(\neg F_1(x))$ or $\exists x(|x|_1 < M' \wedge F_1(x))$ must be true).

Thus, every producible R_θ and L_θ , it can find its necessary counterpart and grow to the target assembly, showing the instance of P_{UAV} is true.

If the instance P_{MAX} is false, $N \neq M$. Assume w.l.o.g. that $N > M$. Consider the assembly R_{N_b} . It is terminal if there does not exist a producible $L_{M_a}, N < M$ or a producible $L_{M_b}, N \leq M$. Since $N > M$, $\exists x(|x|_1 \geq N \wedge F_2(x))$, so none of the necessary assemblies will ever be produced, so R_{N_b} is terminal, and is not A_T . Therefore, the instance of P_{UAV} is false. \square

8.5 Membership

We show membership in $P_{\parallel}^{\text{NP}}$ by giving a deterministic polynomial time algorithm using oracles for problems in NP. We first show a generalized version of the producibility problem is in NP for freezing affinity-strengthening systems. We then show how the solutions to these problems can be used to find terminal assemblies that are not our target assembly.

Definition 17 (Edge-Producibility Problem (eprod)) Given a Bounded Tile Automata System $\Gamma = (\Sigma, \Pi, \Lambda, \Delta, \tau)$, two states $\sigma_L, \sigma_R \in \Sigma$, and an assembly A , does there exist a producible assembly B where the following holds?

1. $|B| < |A|$.
2. $B \neq A$.
3. $A(0) = \sigma_L$.
4. $A(|A| - 1) = \sigma_R$.

Definition 18 (Edge-Transition Terminal Problem (ett)) Given a Bounded Tile Automata System $\Gamma = (\Sigma, \Pi, \Lambda, \Delta, \tau)$, states $\sigma_L, \sigma_R \in \Sigma$, and an assembly A , does there exist a producible assembly B where the following holds?

1. $|B| \leq |A|$.
2. $B \neq A$.
3. $B(0) = \sigma_L$.
4. $B(|A| - 1) = \sigma_R$.
5. B is not a transitional assembly.

Lemma 6 *The edge-producibility and edge-transition terminal problem is in NP for freezing Affinity Strengthening systems.*

Proof In a freezing Affinity Strengthening system $\Gamma = (\Sigma, \Pi, \Lambda, \Delta, \tau)$, the max build sequence length for any producible assembly B is $\mathcal{O}(|B||\Sigma|)$ and we only need to consider assemblies of less than size $|\Lambda|$. These problems can be verified in polynomial time using an assembly B and its build sequence, which allows us to verify that B is

producible. Checking the other conditions can be done in $\mathcal{O}(|A| + |B|)$ time by looking at each tile in A and B . \square

Theorem 12 *Unique Assembly Verification in 1-dimensional freezing Affinity Strengthening Tile Automata is in $P_{\parallel}^{\text{NP}}$.*

Proof Given $\Gamma = (\Sigma, \Pi, \Lambda, \Delta, \tau)$ and a target assembly A , our algorithm uses $2|\Sigma|^2 + 2$ oracles. Two oracle calls check that A is producible, and that Γ is bounded by $|\Lambda|$. Now, we only need to verify that A is terminal, and that A is the only terminal assembly. We create two tables P and T indexed by $\Sigma \times \Sigma$, and then use the remaining $2|\Sigma|^2$ oracle calls to fill out these tables. The value in each cell is calculated using the oracle to solve the edge-producibility and edge-transition terminal problems, $P(\sigma_L, \sigma_R) = \text{eprod}(\Gamma, \sigma_L, \sigma_R, A)$ and $T(\sigma_L, \sigma_R) = \text{ett}(\Gamma, \sigma_L, \sigma_R, A)$ for the two states that index that cell.

Note that in 1-dimensional systems, two assemblies can only attach using a single tile. This means that two assemblies with the same left and rightmost state attach to the same set of assemblies. We use this to help find terminal assemblies. First, we verify that A is a terminal assembly. We may check that A is not transitional by checking each pair of neighboring states. For each state σ' that has affinity on the left side of $A(0)$, we check table P to see if there exists any producible assembly with σ' as its rightmost state. Such an assembly would be able to attach to the left side of A , and thus it is not terminal. We perform the same process for state $A(|A| - 1)$, the rightmost state of A , to find if anything can attach on its right side.

Using the table T , we can find assemblies that are not transitional. If the value at $T(\sigma_L, \sigma_R)$ is 1, the same method as above is used to check if an assembly with those states can attach to another assembly. If it cannot, then this means Γ does not uniquely assemble A .

This algorithm makes a constant number of checks to each cell of the table $\mathcal{O}(|\Sigma|^2)$, and verifying that A is terminal takes $\mathcal{O}(|A|)$ time. The final resulting runtime is $\mathcal{O}(|\Sigma|^2 + |A|)$. \square

9 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 systems 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 three dimensions or allowing a variable temperature. Further limiting Tile Automata may provide more insight into the hardness of these problems.

References

Adleman LM, Cheng Q, Goel A, Huang MDA, Kempe D, de Espanés PM, Rothemund PWK (2002) Combinatorial optimization problems in self-assembly. In: Proceedings of the 34th annual ACM symposium on theory of computing, pp 23–32

Alumbaugh JC, Daymude JJ, Demaine ED, Patitz MJ, Richa AW (2019) Simulation of programmable matter systems using active tile-based self-assembly. In: Thachuk C, Liu Y (eds) DNA computing and molecular programming. Springer, Cham, pp 140–158

Caballero D, Gomez T, Schweller R, Wylie T (2020) Verification and Computation in Restricted Tile Automata. In: Geary C, Patitz MJ (eds) 26th international conference on DNA computing and molecular programming (DNA 26), Leibniz International Proceedings in Informatics (LIPIcs), vol. 174, pp. 10:1–10:18. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany. <https://doi.org/10.4230/LIPIcs.DNA.2020.10>. URL <https://drops.dagstuhl.de/opus/volltexte/2020/12963>

Caballero D, Gomez T, Schweller R, Wylie T (2021) Covert computation in staged self-assembly: Verification is pspace-complete. In: Proceedings of the 29th European Symposium on Algorithms, ESA’21

Cannon S, Demaine ED, Demaine ML, Eisenstat S, Patitz MJ, Schweller RT, Summers SM, Winslow A (2013) 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), Leibniz International Proceedings in Informatics (LIPIcs), vol. 20, pp. 172–184. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik

Cantu AA, Luchsinger A, Schweller R, Wylie T (2020) Covert Computation in Self-Assembled Circuits. *Algorithmica*. <https://doi.org/10.1007/s00453-020-00764-w>. arXiv:1908.06068

Cantu AA, Luchsinger A, Schweller R, Wylie T (2020) Signal Passing Self-Assembly Simulates Tile Automata. In: Y. Cao, S.W. Cheng, M. Li (eds.) 31st International Symposium on Algorithms and Computation (ISAAC 2020), Leibniz International Proceedings in Informatics (LIPIcs), vol. 181, pp. 53:1–53:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany. <https://doi.org/10.4230/LIPIcs.ISAAC.2020.53>. URL <https://drops.dagstuhl.de/opus/volltexte/2020/13397>

Chalk C, Luchsinger A, Martinez E, Schweller R, Winslow A, Wylie T (2018) Freezing simulates non-freezing tile automata. In: International Conference on DNA Computing and Molecular Programming, pp. 155–172. Springer

Chalk C, Luchsinger A, Schweller R, Wylie T (2018) Self-assembly of any shape with constant tile types using high temperature. In: Proceedings of the 26th Annual European Symposium on Algorithms, ESA’18

Cook M (2004) Universality in elementary cellular automata. *Complex Syst* 15(1):1–40

Daymude JJ, Hinnenthal K, Richa AW, Scheideler C (2019) Computing by programmable particles. In: Distributed computing by mobile entities: current research in moving and computing, pp 615–681. Springer, Cham

Demaine ED, Demaine ML, Fekete SP, Ishaque M, Rafalin E, Schweller RT, Souvaine DL (2008) Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. *Natural Comput* 7(3):347–370

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

Doty D, Kari L, Masson B (2013) Negative interactions in irreversible self-assembly. *Algorithmica* 66(1):153–172

Evans C (2014) Crystals that count! physical principles and experimental investigations of dna tile self-assembly. Ph.D. thesis, California Inst. of Tech

Goles E, Meunier PE, Rapaport I, Theyssier G (2011) Communication complexity and intrinsic universality in cellular automata. *Theor Comput Sci* 412(1–2):2–21

Kanaras AG, Wang Z, Bates AD, Cosstick R, Brust M (2003) Towards multistep nanostructure synthesis: programmed enzymatic self-assembly of DNA/gold systems. *Angewandte Chemie International Edition* 42(2):191–194

Kawano R (2018) Synthetic ion channels and DNA logic gates as components of molecular robots. *ChemPhysChem* 19(4):359–366. <https://doi.org/10.1002/cphc.201700982>

Keenan A, Schweller R, Sherman M, Zhong X (2016) Fast arithmetic in algorithmic self-assembly. *Natural Comput* 15(1):115–128

Kimna C, Lieleg O (2019) Engineering an orchestrated release avalanche from hydrogels using DNA-nanotechnology. *J Control Release*. <https://doi.org/10.1016/j.jconrel.2019.04.028>

Kuroda SY (1964) Classes of languages and linear-bounded automata. *Information and Control* 7(2):207–223. [https://doi.org/10.1016/S0019-9958\(64\)90120-2](https://doi.org/10.1016/S0019-9958(64)90120-2). URL <http://www.sciencedirect.com/science/article/pii/S0019995864901202>

Luchsinger A, Schweller R, Wylie T (2018) Self-assembly of shapes at constant scale using repulsive forces. *Natural Comput*. <https://doi.org/10.1007/s11047-018-9707-9>

Padilla JE, Patitz MJ, Pena R, Schweller RT, Seeman NC, Sheline R, Summers SM, Zhong X (2013) Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes. In: Unconventional Computation and Natural Computation, pp. 174–185. Springer

Schweller R, Sherman M (2013) Fuel efficient computation in passive self-assembly. In: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'13, pp. 1513–1525. SIAM

Schweller R, Winslow A, Wyllie T (2017) Complexities for high-temperature two-handed tile self-assembly. In: Brijder R, Qian L (eds) DNA computing and molecular programming. Springer, Cham, pp 98–109

Schweller R, Winslow A, Wyllie T (2019) Nearly constant tile complexity for any shape in two-handed tile assembly. *Algorithmica* 81(8):3114–3135

Schweller R, Winslow A, Wyllie T (2019) Verification in staged tile self-assembly. *Natural Comput* 18(1):107–117

Spakowski H (2006) Completeness for parallel access to np and counting class separation. Ausgezeichnete Informatikdissertationen 2005

Winfree E (1998) Algorithmic self-assembly of DNA. Ph.D. thesis, California Institute of Technology

Winslow A (2015) Staged self-assembly and polyomino context-free grammars. *Natural Comput* 14(2):293–302

Worsch T (2013) Towards intrinsically universal asynchronous ca. *Natural Comput* 12(4):539–550

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.