Covert Computation in Staged Self-Assembly: Verification Is PSPACE-Complete

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

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

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

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

Abstract

Staged self-assembly has proven to be a powerful abstract model of self-assembly by modeling laboratory techniques where several nanoscale systems are allowed to assemble separately and then be mixed at a later stage. A fundamental problem in self-assembly is Unique Assembly Verification (UAV), which asks whether a single final assembly is uniquely constructed. This has previously been shown to be Π_2^p-hard in staged self-assembly with a constant number of stages, but a more precise complexity classification was left open related to the polynomial hierarchy.

Covert Computation was recently introduced as a way to compute a function while hiding the input to that function for self-assembly systems. These Tile Assembly Computers (TACs), in a growth only negative aTAM system, can compute arbitrary circuits, which proves UAV is coNP-hard in that model. Here, we show that the staged assembly model is capable of covert computation using only 3 stages. We then utilize this construction to show UAV with only 3 stages is Π_2^p-hard. We then extend this technique to open problems and prove that general staged UAV is PSPACE-complete. Measuring the complexity of n stage UAV, we show Π_{n-1}^p-hardness. We finish by showing a Π_{n+1}^p algorithm to solve n stage UAV leaving only a constant gap between membership and hardness.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Problems, reductions and completeness

Keywords and phrases self-assembly, covert computation, staged self-assembly, assembly verification

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.23

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

1 Introduction

The *Staged Self-Assembly* model was designed as an extension to the standard hierarchical model of tile self assembly that mimics the abilities of scientists in the lab to control the assembly process by mixing test tubes. The additional features in this model allow for more efficient tile complexity, but increased complexity of certain verification problems.

We use the concept of *Covert Computation*, a requirement of a computational system stipulating that the input and computational history of the computation be hidden in the final output of the system, within the context of Staged Self-assembly, an extension to tile self-assembly that allows for basic operations such as mixing self-assembly batches over a sequence of distinct stages. We use this connection to resolve open questions regarding the complexity of the *Unique Assembly Verification* (UAV) problem within staged self-assembly- the problem of whether a given system uniquely produces a specific assembly. The importance of this work stems from the fundamental nature of the UAV problem, along with the natural and experimentally motivated Staged Self-Assembly model. Further, the novel approach by

which our results are obtained, by way of designing Covert Computation systems in Staged Self-Assembly, may be of independent interest as it shows how to utilize Staged Self-Assembly to implement general purpose computing systems with strong guarantees that might be useful for cryptography or have applications for privacy within biomedical computation.

Staged Self-Assembly. The Staged Self-Assembly model [1, 6, 7, 8, 9, 10, 11, 14, 18] is a generalization of the (2-handed) tile assembly model [4] where particles are modeled by 4-sided Wang tiles which nondeterministically combine based on the affinity of tile edges. Tile self-assembly is a well-studied mathematical abstraction used in the study of self-assembly systems with algorithmically complex behavior, and enjoys experimental success through a DNA implementation [19]. In order to add the basic functionality of what an experimentalist with a set of test tubes could execute [17], the staged model extends tile self-assembly by allowing assembly to occur in multiple separate *bins*, and for the contents of these bins to be either combined or split into a new set of bins after each one of a given sequence of *stages*.

Covert Computation. Tile self-assembly can be used as a model of computation in which tiles attach to an input *seed* structure to grow a final output structure encoding the result of the computation. This basic paradigm is one of most promising avenues for the development of nanoscale molecular computing systems (see [19] for recent experimental work using DNA tiles to implement 6-bit circuits). The authors in [5] recently proposed a new constraint on such computing systems termed *Covert Computation*. A covert computation system computes a function with the additional constraint that the output assembly provides no information about either the original input or the computational history, beyond the actual output of the computed function. This is a particularly daunting self-assembly problem since the output is provided in the form of a self-assembled structure that encodes the exact geometric location of every placed tile. In previous methods of tile self-assembly computation, the entire computational history and original input are easily interpreted from the final output assembly. However, while the output assembly specifies the location of each placed tile, the result of the computation can be a function of not just these tile *locations*, but also of the *order* in which these tiles are placed, which is the technique exploited in [5]. This concept provides a useful technique for proving complexity results, and we use it here to show PSPACE-completeness of verifying unique assembly in staged self-assembly.

Unique Assembly Verification. One well-studied problem in tile self-assembly is the Unique Assembly Verification (UAV) problem which asks if a given system uniquely produces a given assembly. This problem was shown to be solvable in polynomial time in the Abstract Tile Assembly Model [2]. The addition of negative interactions and detachment of tiles makes the UAV problem undecidable [12], while *growth-only* systems with no detachments are coNP-complete [5]. The UAV problem in the 2-Handed Assembly Model was first studied in [4] where coNP membership was shown with coNP-completeness in the third dimension. The problem was also shown to be coNP-complete with a variable temperature [15], but constant temperature UAV in the 2HAM is still open. In the staged assembly model, initial investigation in [16] showed coNP-hardness using four stages and Π_2^p-hardness for seven stages. They also showed membership in PSPACE with a conjecture of PSPACE-completeness.

Our Results. In this paper, we introduce the concept of covert computation in the context of staged self-assembly for the purpose of establishing the complexity of unique assembly verification within the model. First, we show that staged self-assembly is capable of covert

Table 1 Complexities of Unique Assembly Verification in the Staged Assembly Model with respect to the number of stages n. Our results are in bold. *This result uses the temperature as an input parameter/variable for the problem. All other results are true even with a constant temperature.

Stages	Membership		Hardness	
1 (2HAM)	coNP	In [4]	coNP-complete*	In [15]
2	$\mathbf{\Pi_3^p}$	**Thm. 19**	coNP-hard*	In [15]
3	$\mathbf{\Pi_4^p}$	**Thm. 19**	$\mathbf{\Pi_2^p}$**-hard**	**Thm. 6**
$n > 3$	$\mathbf{\Pi_{n+1}^p}$	**Thm. 19**	$\mathbf{\Pi_{n-1}^p}$**-hard**	**Thm. 12**
General	PSPACE	In. [16]	**PSPACE-complete**	**Thm. 10**

computation even when limited to three stages. Next, we use this fact to show UAV is PSPACE-complete in staged self-assembly, resolving the open problem from [16]. Along the way, we improve on some results from [16]: we show that UAV is Π_2^p-hard with just three stages, improving on the previous hardness result requiring seven stages. We then generalize this result to show that for n stages, UAV is Π_{n-1}^p-hard, but yields a Π_{n+1}^p algorithm, leaving only a gap of two in levels between membership and hardness for this problem. An overview of our results and known results related to UAV is shown in Table 1.

2 Preliminaries

We first provide definitions for the staged self-assembly model and covert computation.

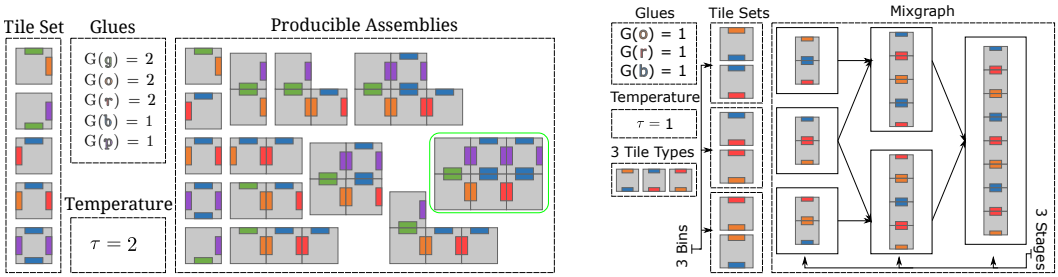
Tiles. A *tile* is a non-rotatable unit square with each edge labeled with a *glue* from a set Σ. Each pair of glues $g_1, g_2 \in \Sigma$ has a non-negative integer *strength* $\text{str}(g_1, g_2)$.

Configurations, bond graphs, and stability. A *configuration* is a partial function $A : \mathbb{Z}^2 \to T$ for some set of tiles T, i.e. an arrangement of tiles on a square grid. For a given configuration A, define the *bond graph* G_A to be the weighted grid graph in which each element of $\text{dom}(A)$ is a vertex, and the weight of the edge between a pair of tiles is equal to the strength of the coincident glue pair. A configuration is said to be τ*-stable* for positive integer τ if every edge cut of G_A has strength at least τ, and is τ*-unstable* otherwise.

Assemblies. For a configuration A and vector $\vec{u} = \langle u_x, u_y \rangle$ with $u_x, u_y \in \mathbb{Z}^2$, $A + \vec{u}$ denotes the configuration $A \circ f$, where $f(x, y) = (x + u_x, y + u_y)$. For two configurations A and B, B is a *translation* of A, written $B \simeq A$, provided that $B = A + \vec{u}$ for some vector $\vec{u}$. For a configuration A, the *assembly* of A is the set $\tilde{A} = \{B : B \simeq A\}$. An assembly $\tilde{A}$ is a *subassembly* of an assembly $\tilde{B}$, denoted $\tilde{A} \sqsubseteq \tilde{B}$, provided that there exists an $A \in \tilde{A}$ and $B \in \tilde{B}$ such that $A \subseteq B$. An assembly is τ*-stable* provided the configurations it contains are τ-stable. Assemblies $\tilde{A}$ and $\tilde{B}$ are τ*-combinable* into an assembly $\tilde{C}$ provided there exist $A \in \tilde{A}$, $B \in \tilde{B}$, and $C \in \tilde{C}$ such that $A \cup B = C$, $A \cap B = \varnothing$, and $\tilde{C}$ is τ-stable.

Two-handed assembly and bins. We define the assembly process in terms of bins. A *bin* is an ordered tuple (S, τ) where S is a set of *initial* assemblies and τ is a positive integer parameter called the *temperature*. For a bin (S, τ), the set of *produced* assemblies $P'_{(S,\tau)}$ is defined recursively as follows:

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



(a) 2HAM Example. **(b)** Staged Self-Assembly Example.

Figure 1 (a) A 2HAM example that uniquely builds a 2×3 rectangle. The top 4 tiles in the tile set all combine with strength-2 glues building the 'L' shape. The tile with blue and purple glues needs two tiles to cooperatively bind to the assembly with strength 2. All possible producibles are shown with the terminal assembly highlighted. (b) A simple staged self-assembly example. The system has 3 bins and 3 stages, as shown in the mixgraph. There are three tile types in our system that we assign to bins as desired. From each stage only the terminal assemblies are added to the next stage. The result of this system is the assembly shown in the bin in stage 3.

A produced assembly is *terminal* provided it is not τ-combinable with any other producible assembly, and the set of all terminal assemblies of a bin (S, τ) is denoted $P_{(S,\tau)}$. Intuitively, $P'_{(S,\tau)}$ represents the set of all possible assemblies that can self-assemble from the initial set S, whereas $P_{(S,\tau)}$ represents only the set of supertiles that cannot grow any further. The assemblies in $P_{(S,\tau)}$ are *uniquely produced* iff for each $x \in P'_{(S,\tau)}$ there exists a corresponding $y \in P_{(S,\tau)}$ such that $x \sqsubseteq y$. Thus unique production implies that every producible assembly can be repeatedly combined with others to form an assembly in $P_{(S,\tau)}$.

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

Given a staged assembly system, for each $1 \leq i \leq r$, $1 \leq j \leq b$, we define a corresponding bin $(R_{i,j}, \tau)$ where $R_{i,j}$ is defined as follows:

1. $R_{1,j} = T_j$ (this is a bin in the first stage);
2. For $i \geq 2$, $R_{i,j} = \left(\bigcup_{k:\ (m_{i-1,k}, m_{i,j}) \in M_{r,b}} P_{(R_{(i-1,k)},\tau)} \right)$.

Thus, the j^{th} bin in stage 1 is provided with the initial tile set T_j, and each bin in any subsequent stage receives an initial set of assemblies consisting of the terminally produced assemblies from a subset of the bins in the previous stage as indicated by the edges of the mix graph.[1] The *output* of the staged system is the union of all terminal assemblies from each of the bins in the final stage.[2] We say this set of output assemblies is *uniquely produced* if each bin in the staged system uniquely produces its respective set of terminal assemblies.

[1] The original staged model [9] only considered $\mathcal{O}(1)$ distinct tile types, and thus for simplicity allowed tiles to be added at any stage. Since our systems may have super-constant tile complexity, we restrict tiles to only be added at the initial stage.

[2] This is a slight modification of the original staged model [9] in that the final stage may have multiple bins. However, all of our results apply to both variants of the model.

Covert Computation. Tile assembly computers were first defined in [5, 13] with respect to the aTAM. We provide formal definitions of both Tile Assembly Computers and Covert Computation with respect to the Staged Self-Assembly model.

A Staged Tile Assembly Computer (STAC) for a function f consists of a staged self-assembly system, and a format for encoding the input into tiles sets and a format for reading the output from the terminal assembly. The input format is a specification for what set of tiles to add to a specific bin in the first stage. Each bit of the input must be mapped to one of two sets of tiles for the respective bit position: a tile set representing "0", or tile set representing "1". The input set for the entire string is the union of all these tile sets. Our staged self assembly system, with the set of tiles needed to build the input seed added in a designated bin, is our final system which performs the computation. The output of the computation is the terminal assembly the system assembles. To interpret what bit-string is represented by the assembly, a second *output* format specifies a pair of sub-assemblies and locations for each bit. An assembly that represents a bitstring is created by the union of each sub-assembly represented by each bit.

For a STAC to *covertly* compute f, the STAC must compute f and produce a unique assembly for each possible output of f. Thus, for all x such that $f(x) = y$, a covert STAC that computes f produces the same output assembly representing output y for each possible input x, making it impossible to determine which input value x was provided to the system.

Input Template. An n-bit input template over tile set T is a sequence of ordered pairs of tile sets over T: $I = (I_{0,0}, I_{0,1}), \ldots, (I_{n-1,0}, I_{n-1,1})$. For a given n-bit string $b = b_0, \ldots, b_{n-1}$ and n-bit input template I, the input tile set for b with respect to I is the set $I(b) = \bigcup_i I_{i,b_i}$.

Output Template. An n-bit output template over tile set T is a sequence of ordered pairs of configurations over T: $O = (C_{0,0}, C_{0,1}), \ldots, (C_{n-1,0}, C_{n-1,1})$. For a given n-bit string $x = x_0, \ldots, x_{n-1}$ and n-bit output template O, the *representation* of x with respect to O is $O(x) =$ the assembly of $\bigcup_i C_{i,x_i}$. A template is valid for a temperature parameter $\tau \in \mathbb{Z}^+$ if this union never contains overlaps for any choice of x, and is always τ-stable. An assembly $B \supseteq O(x)$, which contains $O(x)$ as a subassembly, is said to represent x as long as $O(d) \not\subseteq B$ for any $d \neq x$.

Function Computing Problem. A *staged tile assembly computer* (STAC) is an ordered triple $\Im = (\Gamma, I, O)$ where $\Gamma = (M, \{\varnothing, T_2, \ldots, T_i\}, \tau)$ is a staged self assembly system, I is an n-bit input template, and O is a k-bit output template. A STAC is said to compute function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^k$ if for any $x \in \mathbb{Z}_2^n$ and $y \in \mathbb{Z}_2^k$ such that $f(x) = y$, then the staged self assembly system $\Gamma_{\Im,x} = (M, \{I(x), T_2, \ldots, T_i\}, \tau)$ uniquely assembles a set of assemblies which all represent y with respect to template O.

Covert Computation. A STAC *covertly* computes a function $f(x) = y$ if 1) it computes f, and 2) for each y, there exists a unique assembly A_y such that for all x, where $f(x) = y$, the system $\Gamma_{\Im,x} = (M, \{I(x), T_1, \ldots, T_i\}, \tau)$ uniquely produces A_y. In other words, A_y is determined by y, and every x where $f(x) = y$ has the exact same final assembly.

3 Covert Computation in Staged Self-assembly

Here, we demonstrate covert computation in the staged assembly model. This construction creates a logic circuit using a 3-stage temperature-2 system with a number of bins polynomial in the size of the circuit. We consider only circuits made up of functionally universal NAND gates, but these techniques could be used to create any 2-input gate.

Figure 2b shows a basic overview of the mixgraph used for the covert computation implementation. The method requires three stages with a polynomial number of mixing bins.

- In the first stage, we assemble the components needed to perform the computation. These include an *Input Assembly*, which encodes the input to the function, *Gate Assemblies*, which act as individual gates and perform the computation via their attachment rules and geometry, and additional assemblies which are used to help "clean up" our circuit and covertly get the output.
- In stage two, the input assembly and gate assemblies are added to a single bin along with a test tile. The gate assemblies will begin to attach to the input assembly creating a *Circuit Assembly*. Once the computation is complete, the test tile can attach to the circuit assembly if and only if the output is true. The circuit assembly is terminal in this bin and will be passed to the final stage.
- The final stage adds additional assemblies to the bin along with most of the tile set as single tiles (not shown in figure). The additional assemblies read the output of the circuit and it grows into one of the output templates. The *Output Frame* searches for the test tile representing the output of the circuit. The single tiles fill in any spaces left in the circuit assembly that would show the computation history, thereby turning the assembly into the output template. This requires a linear number of additional bins in the first and second stage to store these single tiles while mixing takes place in other bins.

For our circuit assembly we implement Planar Logic Circuits with only NAND gates. An example circuit and an assembly showing how the gates are laid out are shown in Figure 2a. Wires are represented by 2×3 blocks of tiles shown in blue in the image. Input and Gate assemblies contain a subset of the tiles in each block we call *arms* which represent the values being passed along the wires. The input assembly is a comb-like structure that is designed so that each input bit reaches the gate it is used at (Figure 3a). For each NAND gate in the circuit we have 4 different assemblies, one for each possible input to the gate. A gate assembly can cooperatively bind to the input assembly if the variable values match. The gate assembly has a third arm that represents the output. This allows the next gate assembly to attach, which continues propagating until the computation is done and the circuit assembly is complete. We now cover the construction in detail by stage.

3.1 First Stage - Assembly Construction

Each bin in the first stage will individually create the assemblies that will come together in the next stage. For an n-input k-gate NAND logic circuit (considering crossovers as three XOR gates [5]), we have an input assembly, $4k$ gate assemblies, and a constant number of other assemblies that will be used in the final stage. Here we will describe the details of the individual assemblies created in addition to the *arms*, which function as wires in our system.

Input. For each bit of the input we have two possible input bit assemblies (Figure 3a). The value of the bit determines which tiles will be added to create that input bit assembly in the first stage. Figure 3a shows the selected assemblies that come together to form the *input assembly* shown in Figure 3b. Each subassembly has a domino which we call an "arm" representing the corresponding bit value. The shape of these assemblies depends on the gates to which they input because the arm of the assembly must reach the location of the gate it inputs to. The last input bit assembly also contains an extra set of tiles that reach the final output gate with a strength-1 glue on its north end and two glues on it's east to allow for the test tile and output frame to attach.

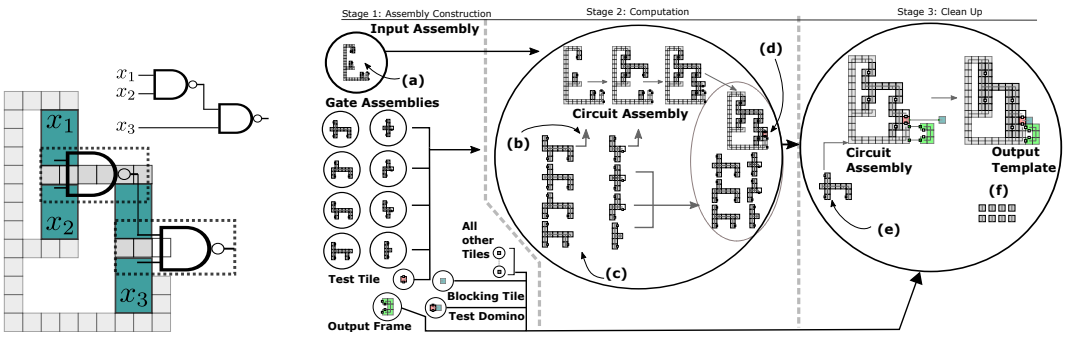


Figure 2 (a) Simple 3-input logic circuit using 2 NAND gates, and the high-level abstraction of the circuit assembly showing the input variables and gates highlighted as blocks. Blue blocks are the sections of the assemblies we call *Arms* that function as wires in the systems. (b) (1) Our input assembly and gate assemblies are constructed in separate bins. (2) Gate assemblies attach to the input assembly forming a circuit assembly. (3) Unused gates are terminal in the second stage. (4) This circuit evaluates to true, so the test tile will be able to attach. (5) Gate assemblies in this stage grow into a circuit using single tiles. (6) Single tiles fill in open spots in the circuit assembly to hide the history. The additional assemblies are used to reach the output template.

Arms. We describe assemblies as having input or output *arms* which function as the wires of our circuit. Arms are vertical dominoes that represent bit values, with their location on the assembly representing the bit having a value 0 or 1 (Figure 4a). The output arm being in the left position represents a bit value of 0, with the right position representing 1. The locations of input arms are complementary (right represents 0, left represents 1) to the output arms. These arms have a glue on the second tile on the inner side. An input arm will attach to an output arm to "read" the bit (Figure 4b) if they represent the same wire and the same value. This glue is a strength-1 glue, so the assembly must attach cooperatively elsewhere in the assembly. Another key feature of these arms is the ability to hide the information passed through by adding single tiles in a later stage. The spaces left by the attachment may be filled by single tiles which results in an assembly which looks like Figure 4c where the value passed cannot be read. This feature will be used in the final stage of our system.

Gates. For each gate we create four assemblies with each representing one of the valid input/output combinations of the desired logic gate. Each gate assembly has two input arms and one output arm. We encode the logic gate by placing the output arm in the column representing the output of the gate when input with the bits represented by the input arms (Figure 3c). This assembly has strength-1 glues on each of its arms. The shape of each gate is dependent on the layout of the circuit since the output arm needs to reach to the next gate. In the case a gate has a fan out (outputs to multiple gates) a gate assembly may have multiple output arms which share arm position. We will refer to the final gate of the circuit as the *output gate*. It does not contain an output arm but instead contains a flag tile to represent an output of false, or no flag tile to represent an output of true which can be seen in Figure 3d. The flag tile also contains a strength-1 glue on it's south edge which allows for the test tile to attach.

3.2 Second Stage - Computation

In the second stage there is a single bin where the circuit assembly is created. In this stage the input assembly and the gate assemblies are mixed together to compute the encoded circuit. The computation starts by attaching gates to the input assembly to begin to build

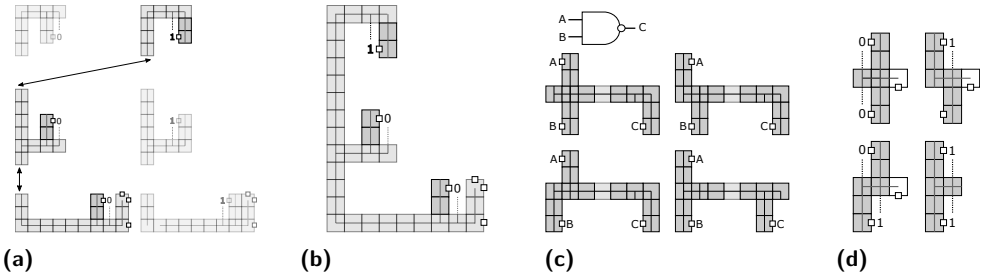


Figure 3 (a) Possible input bit assemblies for a 3 bit function. Solid lines between tiles indicate a strength 2 glue between the tiles. Small boxes indicate a strength 1 glue. For each bit we select either the left or the right assembly based on the value of the bit and add those tiles to our input bin in the *First Stage*. Lighter tiles are not used. (b) The input assembly that is constructed in the *First Stage*. The last input bit assembly contains an extra column of tiles that reaches to where the output gate will be for cooperative attachment of the test tile. (c) 4 gate assemblies, one for each possible input combination of a NAND gate. Glues are labeled to match the wires of the NAND gate. (d) Output gates. True output gates contain a flag tile (white).

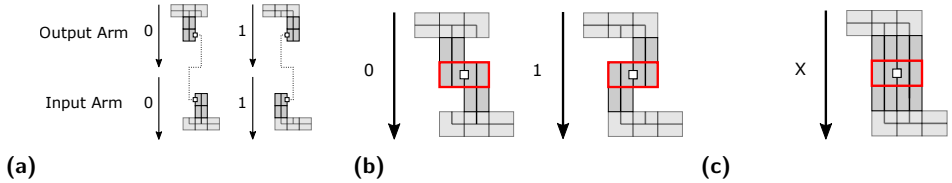


Figure 4 (a) Information being passed along a wire is represented by the position of a domino called an arm. Output arms represent a signal of "1" of "0" by being in the left or right position, respectively. Input arms read bit values and have complimentary arm placement to allow for attachment. (b) Information is passed by attachment. Another assembly may attach if the arms have matching glues (they represent the same wire) and they have complementary arms (represent the same bit value). (c) In the final stage we add additional tiles to hide the information that was passed along a wire.

the circuit assembly. Once both inputs to a gate are present on the circuit assembly, the next gate assembly can cooperatively attach to the circuit assembly since each arm has an attachment strength of 1 as seen in Figure 5a. In this stage we also add the test tile. If the output of the circuit is true, the flag tile can attach as in Figure 5c. If the output is false, the terminal circuit assembly can be seen in Figure 5b. The test assembly is not able to attach to the circuit assembly in this case and will be terminal. We note that at this point it is possible to read the output of the circuit by checking the terminal assemblies, however the computation history can still be read so the covert computation is not complete and we need an additional stage.

3.3 Third Stage - Clean Up

In the third stage we hide the computational history and get the output of the computation. The output template (Full Circuit Assembly) is shown in Figure 5d, which is a circuit assembly with all open spaces in its arm filled in (computation history is hidden) and the additional assemblies attached. The additional assemblies are the Output Frame, which the test tile may attach to, the test domino, which attaches to a circuit assembly but not to the output frame, the blocking tile, which turns a test tile into a test domino, and all single tiles used in the circuit assembly other than the test tile. The difference between the true and false output templates is the inclusion/exclusion of the test tile within the Output Frame.

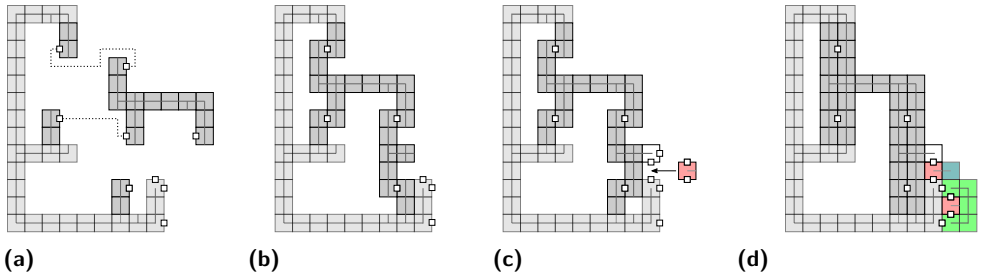


Figure 5 (a) A NAND gate assembly representing input: "10" and output: "1" attaching to the input assembly in the *Second Stage*. (b) False Output Gates which do not contain a flag tile can attach to the circuit if the output is false. This assembly is terminal in the second bin. (c) True Output Gates have an additional flag tile (white) that allows for the test tile (red) to attach cooperatively to the input assembly and the True Output Gate. (d) Single tiles fill in the spaces left by the arms and the output frame attaches forming our target assembly. If there is a flag tile in the output frame the output of this circuit is true. Otherwise, the output is false.

▶ **Theorem 1.** *For any function f computed by an n-input boolean circuit with k gates, there exists a 3-stage $\mathcal{O}(n^2 + k^2)$ bin, temperature-2 staged tile assembly computer that covertly computes f with an output template size of $\mathcal{O}(n^2 + k^2)$.*

Proof. Given any boolean circuit c, we create gate assemblies for each gate. Given the input to this circuit we create input assemblies that encode the input. In the second stage input assemblies start attaching together to form a circuit assembly. Once two inputs to a gate have attached the gate assembly computing the output of that gate is able to attach. Two gate assemblies cannot attach away from the circuit assembly. There only exist strength-1 glues on the outer edges of a gate assembly, thus a gate can only attach to another assembly with a cooperative bind at each arm. This ensures gates only attach once both inputs are present, and forces the circuit to assemble in the correct order. In the second stage we add in the test tile. This test tile may only attach to a circuit assembly that has a flag tile attached. The flag tile is only present on output gates that evaluate to true so the test tile may attach if and only if the circuit evaluates to true. The test tile is terminal otherwise.

In the final bin we add in single tiles to hide the input to the circuit and the inputs to each gate. As explained above each assembly input to this bin will grow into a full circuit assembly. This full circuit assembly will grow into one of our two output frames. The output frame is a full circuit assembly with the output frame attached. The output frame contains a test tile for false and is empty for true. The terminal assembly of our staged system will have a test tile in the output frame if and only if the circuit evaluated to false which is one of our output frames. If the circuit evaluates to true the test tile will not be present.

This system uses a polynomial number of bins in the first and second stage and a single bin the final stage. The number of bins in the first and second stage are bounded by the size of our tile set since we need individual bins to store each tile so they do not combine before the final stage. The max size of a gate assembly is $\mathcal{O}(max(n, k))$ since in the worst case a gate needs to stretch across the whole circuit. The same bound applies to input bit assemblies. Therefore the size of the system (the number of tile types + the size of the mix graph) is $\mathcal{O}(n^2 + k^2)$ ◀

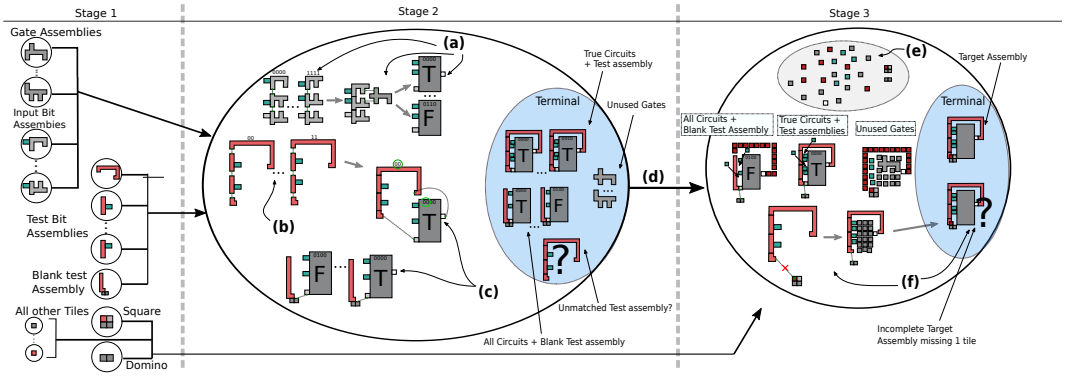


Figure 6 A high level overview of the staged system created from an instance of ∀∃SAT. (a) An input assembly is created for every possible input to ϕ and is evaluated using the computation technique from Section 3. (b) A test assembly is created for every possible input to X. (c) Test assemblies can attach to a true circuit assembly with the same assignment to X. Blank test assemblies attach to any circuit. (d) Terminal assemblies are passed to the next stage, including unmatched test assemblies if any exist. (e) In this stage we add the domino and square assemblies, as well as every other single tile of the target assembly. (f) Any unmatched test assembly will grow into an incomplete target assembly since it cannot attach to the square assembly. These incomplete target assemblies are terminal, meaning the UAV instance is false.

4 Unique Assembly Verification

We now utilize covert computation to show that the open problem of Unique Assembly Verification in staged self assembly is PSPACE-complete. We start by showing UAV with 3 stages is Π_2^p-hard. We then show how to extend this construction to show that general staged UAV is PSPACE-complete. With some adjustments the same concept is used to show that when limiting the system to n stages, the problem of UAV is Π_{n-1}^p-hard.

▶ **Problem 2** (Staged Unique Assembly Verification). *Given a staged system* Γ *and an assembly* A*, does* Γ *uniquely assemble* A*?*

4.1 3-stage UAV is Π_2^p-hard

We modify the covert computation construction to provide a reduction from ∀∃SAT. Given an instance of ∀∃SAT, we create a 3-stage temperature-2 staged system that uniquely produces a target assembly iff the given instance of ∀∃SAT is true. The reduction uses the same high-level idea as [16] and [3]. The process begins with the construction of an assembly for every input to the ∀∃SAT formula. Circuit assemblies build from these inputs and are flagged as true or false, while encoding a partial assignment through their geometry. Separate "test" assemblies are constructed that also encode a partial assignment to the same variables, which attach to true circuit assemblies with matching assignments. The systems uniquely assembles a target assembly if for all test assemblies there exists a compatible true circuit assembly for it to attach to. See Figure 6 for a visual overview of the created system.

▶ **Problem 3** (∀∃SAT). *Given an* n*-bit boolean formula* $\phi(x_1, x_2 ... x_n)$ *with the inputs divided into two sets* X *and* Y*, for every assignment to* X*, does there exist an assignment to* Y *such that* $\phi(X, Y) = 1$*?*

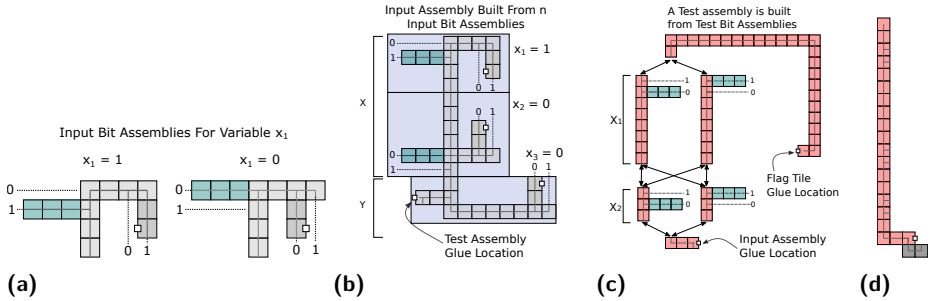


Figure 7 (a) Input bit assemblies for variables in X with geometry on the left reflecting the bit value. (b) An example initial circuit assembly for input $x_1 = 1, x_2 = 0, x_3 = 0$. The geometry on the left side of the assembly represents the assignment of X. (c) Separately built test bit assemblies nondeterministically attach to build one test assembly for every assignment to variables in X. (d) The blank test assembly is composed of the same base but has no protruding arms.

4.1.1 First Stage

Input and Gate Assemblies. In the first stage an *input bit assembly* for both assignments to every variable $x_1, \ldots, x_n$ is built in its own bin ($2n$ bins in total). Input bit assemblies have the same structure as in the covert computation construction, except that input bit assemblies representing bits in X also have a horizontal row of tiles on the left of the frame that reflects the bit value. Figure 7a shows this modification to the input bit assemblies. The bit assemblies representing variable x_n no longer has additional tiles that attach to the test tile used in section 3. The input bit assemblies representing variable $x_{|X|+1}$ have an additional 2 tiles attached, which are used to attach to the test assembly. Gate Assemblies are built in the same way described in Section 3.

Test Assemblies. Similar to the input bit assemblies, two *test bit assemblies* are constructed for every variable in X. A test bit assembly is a column of connected tiles, with a horizontal row of 3 tiles extending to the right, the position of this row represents an assignment "0" or "1". An example test assembly building from separate test bit assemblies is shown in Figure 7c. A test assembly is composed of $|X|$ test bit assemblies. Test assemblies have additional geometry that allow them to attach to a circuit assembly.

4.1.2 Second Stage

In the second stage the input bit assemblies will attach together nondeterministically to form 2^n unique input assemblies. The "1" and "0" input bit assembly exist for every variable, so the nondeterministic nature of the model allows for the construction of an input assembly for every possible input to the circuit. From this input assembly, computation will begin as described in the covert section. There will exist a circuit assembly for each of the 2^n possible inputs, and each will be flagged as true or false, represented by the existence of a flag tile on the output gate. We call a circuit assembly that contains the flag tile a *true circuit assembly*.

Test Assemblies. Test bit assemblies nondeterministically combine in this stage to create a unique test assembly for every assignment to variables in X, with its assignment encoded in its geometry. A test assembly can cooperatively bind to a true circuit assembly (with the same assignment to X) by having glue strength with the output flag tile and the input assembly (Figure 8b). A test assembly has its assignment encoded in its geometry in a

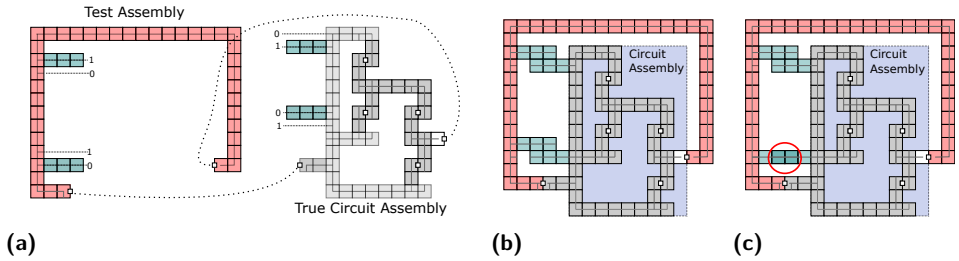


Figure 8 (a) A test assembly (left) and a true circuit assembly that represent the same assignment to variables in X (In this construction true assemblies contain the flag tile). (b) A test assembly attaching to true circuit assembly with a matching assignment to X. (c) A test assembly that is geometrically blocked from attaching to a true circuit assembly due to having a different assignment to X. The red circled area shows the point of overlap.

complementary fashion to that of a circuit assembly. This ensures that a test assembly is geometrically blocked from attaching to a circuit assembly that encodes a different assignment to X (Figure 8c). If there are no true circuit assemblies with the assignment x to X, the test assembly that represents that assignment of x will be terminal in the second stage. We refer to these as *unmatched* test assemblies.

▶ **Lemma 4.** *Let t_x be the test assembly representing the assignment x to the variables in X. t_x is unmatched (terminal in the second stage) if and only if for all assignments y to the variables in Y ($\phi(x, y) = 0$).*

Proof. True circuit assemblies are the only assemblies which have the necessary glue types to attach to t_x. The remaining question is whether a true circuit assembly exists which represents a compatible assignment. t_x can attach to any true circuit assembly with a matching assignment to X, regardless of that circuit's assignment to Y. It follows that if there exists an assignment y to Y such that $\phi(x, y) = 1$, then t_x is not terminal. The negation of which is $\forall y(\phi(x, y) = 0)$, then t_x is terminal. ◀

4.1.3 Third Stage

The third stage utilizes a single bin that all assemblies are combined in. Nearly all single tiles of the target assembly are added. Four single tiles are specifically excluded, and instead two subassemblies are added in. This is done carefully to ensure the following property: every assembly except unmatched test assemblies from the second stage will grow to the target assembly. Our target assembly contains a circuit assembly attached to a test assembly with every empty spot filled in. At the point where a test assembly attaches a the circuit assembly, a domino assembly is attached completing the target assembly as seen in Figure 9a.

▶ **Lemma 5.** *Let A be the set of initial assemblies in the sole bin in the third stage. For all assemblies $a \in A$, a will grow to the target assembly iff a is not an unmatched test assembly.*

Proof. All individual tiles of the target assembly are added into the last stage, with the exception of four withheld tiles: the two tiles where the test assembly and input assembly meet, and the two tiles below that (tiles A, B, C, D in Fig. 9a). Instead of these four tiles, two assemblies are added that we refer to as the square and domino (Fig. 9b). These two assemblies perform the function of allowing every initial assembly besides unmatched test assemblies to grow into the target assembly. True circuit assemblies with test assemblies attached will have their empty spaces filled by single tiles, and the domino assembly will

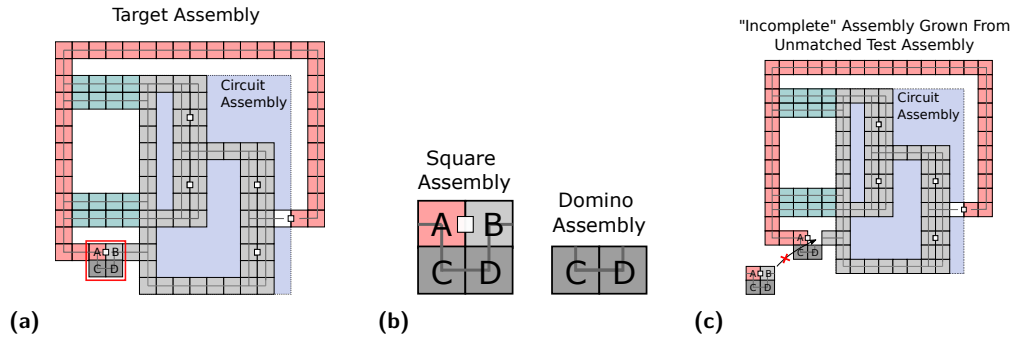


Figure 9 (a) An example target assembly. The area boxed in red shows where the test assembly meets the input assembly (A, B), and the adjacent domino (C, D). The four tile types A, B, C, D are not added in individually at the third stage. (b) The two additional assemblies that are added in at the third stage, composed of the same four tile types. (c) The square assembly is geometrically blocked from attaching to an assembly that grew from an unmatched test assembly, as they both contain tile A.

attach. Unused gates will grow to a near-complete circuit, attach to the square assembly, and then continue to grow to the target assembly. True and False Circuit Assemblies with blank test assemblies attached already contain the four withheld tiles, so will grow to the target by attaching to all necessary single tiles. Unmatched test assemblies that did not attach to a true circuit assembly can grow to a near complete target assembly, however, it will never acquire tile B (Fig. 9a), as it could only achieve this by attaching to the square assembly. They both contain tile A, making it geometrically blocked from doing so (Fig. 9c). ◀

▶ **Theorem 6.** *UAV in the Staged Assembly Model with three stages is* Π_2^p*-hard with* $\tau = 2$.

Proof. Given an instance of ∀∃SAT, the reduction provides an instance of a 3-stage temperature-2 UAV instance which is true if and only if the instance of ∀∃SAT is true.

If the instance of ∀∃SAT is true, then for all assignments x to X, there exists an assignment y to Y with $\phi(x, y) = 1$. By Lemma 4, this implies there will be no unmatched test assemblies. By Lemma 5, every assembly that is not an unmatched test assembly or grown from an unmatched test assembly will grow into the target assembly in the third stage. Thus, the system uniquely produces the target assembly. If the ∀∃SAT instance is false, then there exists an assignment x to X, s.t. for all assignments y to Y, $\phi(x, y) = 0$. By Lemma 4, a test assembly representing assignment x would be unmatched, and by Lemma 5, unable to grow into the target assembly. Thus, this UAV instance is false. ◀

4.2 Staged UAV is PSPACE-hard

In this section, we explain at a high level how the reduction is extended to reduce from TQBF with n quantifiers over $n$variables to temperature-2 $\mathcal{O}(n)$-stage UAV, showing that Staged UAV is PSPACE-Hard.

▶ **Problem 7** (*TQBF*). *Given a boolean formula* ϕ *with* n *variables* $x_1, \ldots, x_n$*, is it true that* $\forall x_1 \exists x_2 \ldots \forall x_n (\phi(x_1, \ldots, x_n) = 1)$*?*

We utilize the same technique used in section 4.1 which reduced from ∀∃SAT, a special case of $TQBF$ limited to only 2 quantifiers, but adapt the technique to work with a QBF with n quantifiers $\forall x_1 \exists x_2 \ldots \forall x_n (\phi(x_1, \ldots, x_n) = 1)$. In the 3rd stage, instead of adding

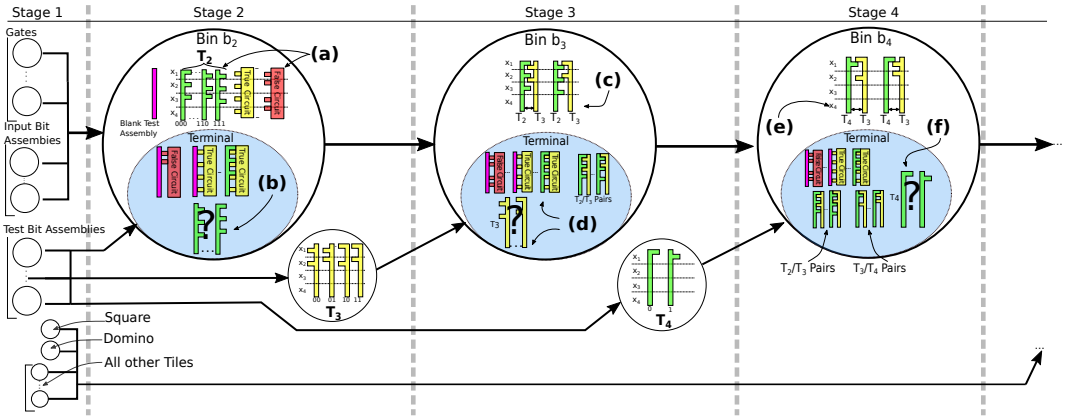


Figure 10 An example mix graph for an instance of TQBF with 4 variables. (a) Test bit assemblies combine into T_2 test assemblies. Circuit assemblies evaluate every input. (b) T_2 test assemblies attach to compatible (matching partial assignment) true circuits. Any unmatched T_2 assemblies are passed to the next stage. (c) T_3 test assemblies are added in and attach to compatible T_2 test assemblies. (d) Any unmatched T_3 assemblies are passed to the next stage. (e) T_4 test assemblies are added and attach to compatible T_3 test assemblies. (f) The existence of an unmatched T_4 assembly directly corresponds to the truth of the TQBF instance.

in single tiles to "clean up", we add in a second set of test assemblies that represent an assignment with one less variable in the next stage and are complementary in their geometry. These new test assemblies then attach to previous test assemblies that were terminal in the previous stage with matching partial assignments. This process computes an additional quantifier. We can then repeat this process of adding in complementary sets of test assemblies for the number of quantifiers required. In the final stage, if a test assembly from the final set couldn't find a complementary test assembly to attach to, the instance of TQBF is false, and that test assembly is prevented from growing to the target assembly. This allows the truth of instance of staged UAV to correspond to the truth of the QBF. See Figure 10 for a depiction of the mix graph. We now show how in a certain stage the existence of a terminal test assembly relates to the truth of a statement about the boolean formula.

In total the system will have $n+1$ stages, and $n-1$ sets of test assemblies will be added (denoted $T_2, \ldots, T_n$). The set T_s will be mixed in at stage s. The first set T_2 represents an assignment to $x_1, \ldots, x_{n-1}$, and each consecutive set represents one less variable than the set before it, i.e., a test assembly $t_s \in T_s$ represents a partial assignment to $x_1, \ldots, x_{n-s-1}$. The sets alternate between type L and R, which correlates to the direction the arms face (Compare T_3 and T_4 in Figure 10). We build all these sets of test assemblies using the same method in the first stage, and pass them along through "helper" bins until they are needed.

▶ **Lemma 8.** *Let $TERM(A, b) \iff$ (Assembly A is terminal in bin b). Let a be the number of variables the test assemblies in T_s represent ($a = n - s + 1$). Let $t_s(x_1, \ldots, x_a)$ be the test assembly $t_s \in T_S$ that represents partial assignment $x_1, \ldots, x_a$. In the staged system S_P created from an instance of TQBF P over n variables: $\forall s \in \{1, \ldots, n\}(TERM(t_s(x_1, \ldots, x_a), b_s) \iff \forall x_{a+1} \exists x_{a+2}, \ldots, Q x_n(\phi(x_1, \ldots, x_n) = y))$. If s is even, $y = 0$ and $Q = \forall$, and $y = 1$, $Q = \exists$ otherwise.*

▶ **Lemma 9.** *In the staged system S_P created from an instance of TQBF P over n variables, in bin b_{n+1} in stage $n+1$, let A be the set of initial assemblies in b_{n+1}. For all $a \in A$, a will grow to the target assembly if and only if a is not an unmatched test assembly $t_n \in T_n$.*

▶ **Theorem 10.** *Unique Assembly Verification in the Staged Assembly Model is PSPACE-complete with* $\tau = 2$.

Proof. Given an instance of TQBF P over n variables/quantifiers, the reduction provides an instance of $n+1$-stage $\tau = 2$ UAV that is true if and only if P is true. If P is true, then in stage $n+1$, every producible assembly grows into the target assembly. Since n is always even, by Lemma 8, for a bin b_n in stage n, an assembly $t_n \in T_n$ representing an assignment x_1 is terminal in bin b_n if $\forall x_2 \exists x_3, \ldots, \forall X_n(\phi(x_1, \ldots, x_n) = 0)$. If P is true, then the statement $\forall x_1 \exists x_2 \forall x_3, \ldots, \forall X_n(\phi(x_1, \ldots, x_n) = 1)$ is true, and therefore no unmatched $t_n \in T_n$ will be passed into b_{n+1}. By Lemma 9, every initial assembly in b_{n+1} that is not some $t_n \in T_n$ grows into the target assembly. Therefore, the target assembly is uniquely assembled if the instance of TQBF is true. If P is false, then there exists an assignment to x_1 such that $\forall x_2 \exists x_3, \ldots, \forall X_n(\phi(x_1, \ldots, x_n) = 0)$. By Lemma 8, some test assembly $t_n \in T_n$ will be terminal and passed into bin b_{n+1}. By Lemma 9, any $t_n \in T_n$ will not grow into the target assembly. Thus, the instance of staged UAV is false. ◀

4.3 n-Stage Hardness

We now show how the reduction can be used to show hardness for n-stage UAV. We reduce from the boolean satisfiability problem for Π_n^p, which is a quantified boolean formula with n quantifiers (starting with universal) and $n-1$ alternations. We show an instance of Π_n^p-SAT can be reduced to $n+1$-stage $\tau = 3$ UAV.

▶ **Problem 11** ($\Pi_n^p - SAT$). *Given a boolean formula ϕ with variables partitioned into n sets $X_1, \ldots, X_n$, is it true that $\forall X_1 \exists X_2 \ldots Q_n X_n(\phi(X_1, \ldots, X_n))$.*

▶ **Theorem 12.** *For all $n > 1$, UAV in the Staged Assembly Model with n stages is Π_{n-1}^p-hard with $\tau = 2$.*

Proof. The system functions nearly identically to the previous reduction. However, if n is odd, the output gate assemblies will now contain the flag tile if they represent a *false* output, rather than true. Each consecutive test assembly added now represents one less *set* of variables, rather than just one less variable.

If n is even, the system acts in the way previously described. If n is odd, then by Lemma 8 any $t_n \in T_n$ representing an assignment to X_1 is terminal if $\forall X_2 \exists X_3 \ldots \exists X_n(\phi(X_1, \ldots, X_n) = 1)$. However, since we modified the output assemblies to contain the flag tile if they represent a *false* output, they are now terminal if the statement is true for the negation of ϕ. Therefore any t_n representing X_1 is terminal if and only if $\forall X_2 \exists X_3 \ldots \exists X_n(\phi(X_1, \ldots, X_n) = 0)$. In bin b_{n+1} all assemblies besides any t_n grow to the target assembly in the same way. ◀

4.4 UAV Membership

In this section, we improve on previous work and show that an n-stage UAV problem is in Π_{n+1}^p. We use a similar method as [16], by defining three subproblems that are solved as subroutines of a UAV algorithm. However, these subproblems differ from previous work as we make some assumptions about our input. We first define *bounded* bins and systems, then define the three subproblems, and show their complexity. However, due to space constraints, the proofs have been omitted.

▶ **Definition 13** (Bounded). *Given a bin $b = (S, \tau)$ in a staged system where S is the set of initial assemblies and τ is the temperature. Let P_b be the set of producible assemblies in bin b. The bin is* bounded *by an integer $k \in \mathbb{Z}^+$ if for each $a \in P_b, |a| \leq k$. A staged system is bounded if all bins are bounded by some k.*

Table 2 Complexity of these problems in 1 stage (2HAM) and in s stages.

Stages	UAV	BPROD	BTERM	BBIN
1	Π_1^p	Σ_0^p	Π_1^p	Π_1^p
s	Π_{s+1}^p	Σ_s^p	Π_s^p	Π_s^p

Algorithm 1 Staged Unique Assembly Verification Membership Algorithm.

Data: Given a staged system Γ with n stages, and an Assembly A.
Result: Does Γ uniquely assemble A and is Γ bounded?
for *each stage* s' *starting with* $s' = 1$ **do**
  for *each bin* b *in stage* s' **do**
    if *Not* $BBIN_{s'}(\Gamma, |A|, b')$ **then** reject;
for *each bin* b *in stage* n **do**
  if *Not* $BPROD_n(\Gamma, |A|, b, A)$ **then** reject;
  if *Not* $BTERM_n(\Gamma, |A|, b, A)$ **then** reject;
Nondeterministically select an assembly B with $|B| \leq |A|$;
for *each bin* b' *in stage* n **do**
  if $BPROD_n(\Gamma, |A|, b', B)$ **then**
    if $BTERM_n(\Gamma, |A|, b', B)$ **then** reject;
accept;

▶ **Problem 14** (Bounded Producibility (BPROD_s)). *Given a bounded staged system Γ, an integer k (described in unary), a bin b in stage s bounded by k, and an assembly A, is A producible in b?*

▶ **Problem 15** (Bounded Terminal Assembly with producibility promise (BTERM_s)). *Given a bounded staged system Γ, an integer k (described in unary), a bin b in stage s bounded by k, and an assembly $A \in P_b$, is A terminal in b?*

▶ **Problem 16** (Bounded Bin (BBIN_s)). *Given a staged system Γ, a bin b in stage s, an integer k (described in unary), assuming all bins in stages before s are bounded by k, is b bounded by k?*

▶ **Lemma 17.** *For a bin b in stage s of a staged self-assembly system,*
- *the Bounded Producibility problem is in Σ_s^p,*
- *the Bounded Terminal Assembly problem with producibility promise is in Π_s^p, and*
- *the Bounded Bin problem is in Π_s^p*

4.5 UAV_n Membership

We now present a co-nondeterministic algorithm using oracles for the previous problems to solve UAV. For clarity, we use an alternate but equivalent definition of UAV. We provide Algorithm 1 that uses oracles to solve the subproblems presented above.

▶ **Problem 18** (Staged Unique Assembly Verification). *Given a staged tile-assembly system Γ and an assembly A, is Γ bounded by $|A|$, and for each bin in the last stage, is A the only terminal assembly?*

▶ **Theorem 19.** *The n-stage Unique Assembly Verification problem in the staged assembly model is in Π_{n+1}^p.*

5 Conclusion

In this paper we answered an open problem from [16] by showing the Unique Assembly Verification problem in the Staged Self-Assembly Model is PSPACE-complete. To show this, we utilized a construction capable of covert computation and extended it to show Π_2^p-hardness of UAV with three stages. We then extended this reduction to show PSPACE-completeness. This reduction is also used to show Π_{s-1}^p-hardness with s stages.

Several important directions for future work remain open. We use three stages to perform covert computation. Is the 2HAM alone capable of covert computation? If not, what is the lower bound on the number of stages needed? If so, can the construction be used to solve the open problem of UAV in that model? This might also mean fewer stages are needed for our results in the staged model. The two known hardness results for 2HAM utilize either one step into the third dimension or a variable temperature. Perhaps stronger results in the staged assembly model can be obtained with one of these variants.

References

1 Zachary Abel, Nadia Benbernou, Mirela Damian, Erik D. Demaine, Martin L. Demaine, Robin Flatland, Scott D. Kominers, and Robert Schweller. Shape replication through self-assembly and rnase enzymes. In *Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1045–1064, 2010. doi:10.1137/1.9781611973075.85.

2 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.

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

4 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.

5 Angel A. Cantu, Austin Luchsinger, Robert Schweller, and Tim Wylie. Covert Computation in Self-Assembled Circuits. *Algorithmica*, 83:531–552, 2021. arXiv:1908.06068. doi:10.1007/s00453-020-00764-w.

6 Cameron T. Chalk, Eric Martinez, Robert T. Schweller, Luis Vega, Andrew Winslow, and Tim Wylie. Optimal staged self-assembly of general shapes. *Algorithmica*, 80(4):1383–1409, 2018. doi:10.1007/s00453-017-0318-0.

7 Cameron T. Chalk, Eric Martinez, Robert T. Schweller, Luis Vega, Andrew Winslow, and Tim Wylie. Optimal staged self-assembly of linear assemblies. *Natural Computing*, 18(3):527–548, 2019. doi:10.1007/s11047-019-09740-y.

8 Erik Demaine, Matthew Patitz, Robert Schweller, and Scott Summers. Self-assembly of arbitrary shapes using rnase enzymes: Meeting the kolmogorov bound with small scale factor. *Symposium on Theoretical Aspects of Computer Science (STACS2011)*, 9, January 2010. doi:10.4230/LIPIcs.STACS.2011.201.

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 Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Computer Science*, 671:4–18, 2017. Computational Self-Assembly. doi:10.1016/j.tcs.2016.11.020.

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

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

14 Matthew Patitz and Scott Summers. Identifying shapes using self-assembly. *Algorithmica*, 64:481–510, 2012.

15 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.

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

17 Grigory Tikhomirov, Philip Petersen, and Lulu Qian. Fractal assembly of micrometre-scale dna origami arrays with arbitrary patterns. *Nature*, 552, December 2017. doi:10.1038/nature24655.

18 Andrew Winslow. Staged self-assembly and polyomino context-free grammars. *Natural Computing*, 14(2):293–302, 2015. doi:10.1007/s11047-014-9423-z.

19 Damien Woods, David Doty, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. Diverse and robust molecular algorithms using reprogrammable dna self-assembly. *Nature*, 567:366–372, March 2019. doi:10.1038/s41586-019-1014-9.