Unique Assembly Verification in Two-Handed Self-Assembly

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

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

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

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

───── **Abstract** ─────

One of the most fundamental and well-studied problems in tile self-assembly is the Unique Assembly Verification (UAV) problem. This algorithmic problem asks whether a given tile system uniquely assembles a specific assembly. The complexity of this problem in the 2-Handed Assembly Model (2HAM) at a constant temperature is a long-standing open problem since the model was introduced. Previously, only membership in the class coNP was known and that the problem is in P if the temperature is one ($\tau = 1$). The problem is known to be hard for many generalizations of the model, such as allowing one step into the third dimension or allowing the temperature of the system to be a variable, but the most fundamental version has remained open.

In this paper, we prove the UAV problem in the 2HAM is hard even with a small constant temperature ($\tau = 2$), and finally answer the complexity of this problem (open since 2013). Further, this result proves that UAV in the staged self-assembly model is coNP-complete with a single bin and stage (open since 2007), and that UAV in the q-tile model is also coNP-complete (open since 2004). We reduce from Monotone Planar 3-SAT with Neighboring Variable Pairs, a special case of 3SAT recently proven to be NP-hard. We accompany this reduction with a positive result showing that UAV is solvable in polynomial time with the promise that the given target assembly will have a tree-shaped bond graph, i.e., contains no cycles. We provide a $\mathcal{O}(n^5)$ algorithm for UAV on tree-bonded assemblies when the temperature is fixed to 2, and a $\mathcal{O}(n^5 \log \tau)$ time algorithm when the temperature is part of the input.

1 Introduction

Since the inception of tile self-assembly [28], one of the most important algorithmic questions has been determining if a given tile system uniquely self-assembles into a specific assembly structure. This basic algorithmic question, termed the Unique Assembly Verification (UAV) problem, is fundamental for efficiently checking if a designed tile system acts as intended, and is tantamount to the design of an efficient simulator for a tile self-assembly model. Thus, UAV has been a central question for every self-assembly model.

Although many different self-assembly models have been proposed in order to simulate different laboratory or experimental setups, two premiere models have emerged as the primary foci of study. First, is the seeded Abstract Tile Assembly Model (aTAM) [28], in which singleton tiles attach one by one to a growing seed if sufficient bonding strength exists based on glue types of attaching tiles. This model has had many foundational results in recent years showing the limits related to intrinsic universality and program-size complexity [20, 21]. The second model is the hierarchical Two-Handed Tile Assembly Model (2HAM) [9], where any two producible assemblies may be combined (one in each of two hands) to create a new producible assembly provided there is sufficient bonding strength between the two pieces. Many foundational results that are known for the aTAM are still open for the 2HAM.

The 2HAM has been shown to be more powerful than the aTAM in its ability to build infinite fractal patterns [11, 18], its program-size efficiency for finite shapes [8], and its running-time efficiency for the self-assembly of finite shapes [12]. While the aTAM has a polynomial time solution to the UAV problem [2], allowing for the production of efficient simulators [17, 22, 29], the complexity of UAV in the 2HAM has remained a long-standing open problem in the field. The 2HAM appeared formally in 2013 [9], but was essentially defined in staged self-assembly [13] (2007), and a seeded version of the 2HAM appears as the *multiple tile* model in [4] (2004). UAV has been open for all of these models, and our coNP-complete result for UAV in the 2HAM proves that UAV with a single bin and single stage in the staged model is coNP-complete, and that UAV in the multiple tile model is also coNP-complete with polynomial-sized pieces, thus answering both of these long-standing open questions. See [15, 23, 30, 31] for surveys and applications of self-assembly theory.

Previous work on UAV.   A number of results have pushed closer to resolving the complexity of UAV in the 2HAM. One of the first results showed that the simpler problem of determining if a given assembly was at least produced (i.e., built but possibly along with other different assemblies) is polynomial time solvable [16], which serves as a key step in showing that UAV resides within the class coNP. Another result augmented the basic 2HAM model to 3 dimensions and showed coNP-hardness for the 3D 2HAM [9]. A recent result focused on 2D, but allowed the temperature threshold, a parameter that determines how much glue strength is required for assemblies to stick together, to be a variable input to the UAV problem (as opposed to a fixed constant value), and showed coNP-completeness in this scenario [26].

Other approaches considered the allowance of initial assemblies consisting of small prebuilt assemblies, as opposed to only initial singleton tiles, and showed UAV becomes coNP$^{\text{NP}}$-complete with this extension [7]. Alternately, the inclusion of a negative force glue, even without detachments, has also been shown to imply coNP-completeness in the aTAM [10]. Another generalization of the 2HAM allows for up to k hands to create new assemblies, instead of just two, causing the problem to become either coNP-complete or PSPACE-complete, depending on the encoding of the variable k [5]. An even more powerful generalization of the 2HAM is the *staged* model [13], in which multiple distinct stages of self-assembly are considered. Within the staged model, UAV becomes coNP$^{\text{NP}}$-hard after 3 stages, and PSPACE-complete in general [6, 27]. Thus, for nearly every way in which the 2HAM has been extended, a corresponding hardness reduction has been found. Yet, the original question of UAV in the 2HAM has remained open.

Our Contributions.   We show that UAV in the 2HAM is coNP-complete within the original model (2-dimensional, constant bounded temperature parameter, singleton tile initial assemblies), thus resolving the long-standing open problem of UAV in the 2HAM. Further, this

■ **Table 1** Known Results for the Unique Assembly Verification Problem in the 2HAM and the results presented in this paper. $|A|$ is the size of the target assembly, τ is the temperature of the system, and $|T|$ is the number of tile types in the system. Under the **Temperature** column, τ indicates that the temperature may be included as part of the input.

Shape	Dimensions	Temperature	Complexity	Reference
General	2	1	$\mathcal{O}(|A||T|\log|T|)$	[16]
General	3	2	coNP-complete	[9]
General	2	τ	coNP-complete	[26]
General	**2**	**2**	**coNP-complete**	**Thm. 7**
Tree	**2**	τ	$\mathcal{O}(|A|^5 \log \tau)$	**Thm. 12**

proves that UAV in the staged model with a single bin and stage is coNP-complete, and that UAV in the q-tile/multiple-tile model with polynomial-sized pieces is coNP-complete. We augment this result with a positive result for the special case of tree-shaped assemblies, providing a $\mathcal{O}(|A|^5 \log \tau)$ time solution for UAV in this case (where $|A|$ is the size of the assembly) even if τ is included as part of the input.

Our results are highlighted in Table 1 along with other known results for UAV in the 2HAM. To show coNP-hardness for UAV we construct an explicit polynomial-time reduction from Monotone Planar 3-SAT with Neighboring Variable Pairs (MP-3SAT-NVP). This reduction takes inspiration from the recent break-through proof that MP-3SAT-NVP is NP-hard and its use to prove that the connected-assembly-partitioning problem with unit squares is NP-hard [3]. For our tree UAV algorithm, we utilize a cycle decomposition approach over possible produced assemblies combined with dynamic programming.

Overview. The paper is structured as follows. Section 2 formally defines the model, the UAV problem, important definitions, and some small examples. Section 3 has the reduction proving UAV in the 2HAM is coNP-hard. Due to the numerous intricate details related to the proof, the section is broken up into several subsections explaining different aspects of the reduction. Section 4 then gives the algorithms for solving UAV for tree-bonded assemblies. Section 5 then concludes the paper with a summary and future work.

2   Definitions

In this section we overview the basic definitions related to the two-handed self-assembly model and the verification problems under consideration.

Tiles. A *tile* is a non-rotating 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* $\mathrm{str}(g_1, g_2)$.

Configurations. A *configuration* is a partial function $\tilde{A} : \mathbb{Z}^2 \to T$ for some set of tiles T, i.e. an arrangement of tiles on a square grid. For a configuration $\tilde{A}$ and vector $\vec{u} = \langle u_x, u_y \rangle$ with $u_x, u_y \in \mathbb{Z}^2$, $\tilde{A} + \vec{u}$ denotes the configuration $\tilde{A} \circ f$, where $f(x, y) = (x + u_x, y + u_y)$. For two configurations $\tilde{A}$ and $\tilde{B}$, $\tilde{B}$ is a *translation* of $\tilde{A}$, written $\tilde{B} \simeq \tilde{A}$, provided that $\tilde{B} = \tilde{A} + \vec{u}$ for some vector $\vec{u}$.

Bond graphs, and stability. For a given configuration $\tilde{A}$, define the *bond graph* $G_{\tilde{A}}$ to be the weighted grid graph in which each element of $\mathrm{dom}(\tilde{A})$ 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

(a) Cooperative Binding.   **(b)** Bond Graph.   **(c)** Geometric Blocking.
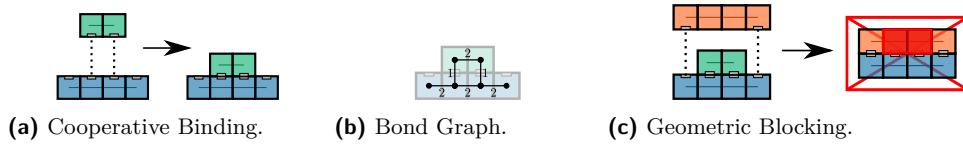
Figure 1 (a) Example of an attachment that takes places using cooperative binding at $\tau = 2$. We denote a glue strength of 1 with a rectangle and a glue of strength 2 with a solid line through the two tiles. Dotted lines between glues indicate that these tiles may attach to each other with the respective strength. Assume assemblies shown are τ-stable unless stated otherwise. (b) The bond graph of the assembly showing that it is τ-stable. (c) These two assemblies are not τ-combinable since this would place two tiles at the same location. We say this is due to geometric blocking.

configuration is said to be τ-*stable* for a positive integer τ if $G_{\tilde{A}}$ is connected and if every edge cut of $G_{\tilde{A}}$ has a weight of at least τ. This means that the sum of the glue strengths along each cut is greater or equal to τ. A small example bond graph is shown in Figure 1b.

Assemblies.   For a configuration $\tilde{A}$, the *assembly* of $\tilde{A}$ is the set $A = \{\tilde{B} : \tilde{B} \simeq \tilde{A}\}$. Informally an assembly A is a set containing all translations of a configuration $\tilde{A}$. An assembly A is a *subassembly* of an assembly B, denoted $A \sqsubseteq B$, provided that there exists an $\tilde{A} \in A$ and $\tilde{B} \in B$ such that $\tilde{A} \subseteq \tilde{B}$. We define $|A|$ to be the number of tiles in a configuration of A.

An assembly is τ-*stable* if the configurations it contains are τ-stable. Assemblies A and B are τ-*combinable* into an assembly C if there exist $\tilde{A} \in A$, $\tilde{B} \in B$, and $\tilde{C} \in C$ such that 1) $\tilde{A} \cup \tilde{B} = \tilde{C}$, 2) $\tilde{A} \cap \tilde{B} = \varnothing$, and 3) $\tilde{C}$ is τ-stable. Informally, two assemblies are τ-combinable if there exist two configurations of the assemblies that may be combined resulting in a τ-stable assembly without placing two tiles in the same location.

Two assemblies combining or binding together is called an attachment. An attachment takes place using *cooperative binding* if the two assemblies do not share a τ-strength glue and instead use multiple weaker glues summing to τ. An example of an attachment that takes place using cooperative binding can be seen in Figure 1a. If an attachment cannot take place because the two tiles would be placed in the same position, it is *geometrically blocked*. Two assemblies whose attachment is geometrically blocked is shown in Figure 1c.

Two-handed Assembly.   A two-handed assembly system (2HAM) is an ordered tuple $\Gamma = (T, \tau)$ where T is a set of tiles and τ is a positive integer parameter called the *temperature*. For a system Γ, the set of *producible* assemblies P'_Γ is defined recursively as follows: 1) $T \subseteq P'_\Gamma$, and 2) If $A, B \in P'_\Gamma$ are τ-combinable into C, then $C \in P'_\Gamma$. We are naturally extending the concept of τ-combinable to single tiles by considering them assemblies of size 1.

A producible assembly is *terminal* provided it is not τ-combinable with any other producible assembly. Denote the set of all terminal assemblies of a system Γ as P_Γ. Intuitively, P'_Γ represents the set of all possible assemblies that can self-assemble from the initial set T, whereas P_Γ represents only the set of assemblies that cannot grow any further. Figure 2 shows a small 2HAM example.

An *Assembly Tree* for a 2HAM system $\Gamma = (T, \tau)$ is any rooted binary tree whose nodes are elements of P'_Γ, the leaves are single tiles from the set T, and the two children of any non-leaf node are τ-combinable into their parent. An assembly tree with root A is said to be an assembly tree for assembly A. A small example is shown in Figure 2c.

(a) UAV Instance. **(b)** Producible Assemblies. **(c)** Small Assembly Tree.
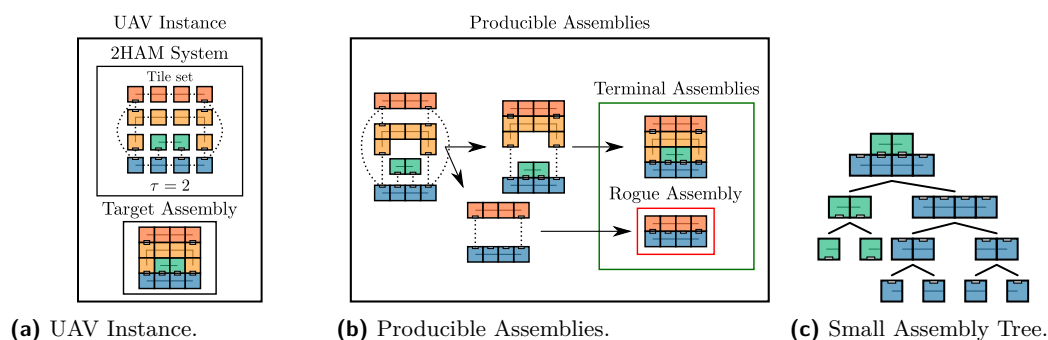
■ **Figure 2** (a) An example instance of the Unique Assembly verification problem. The input is the 2HAM system (tile set and temperature) and the target assembly. (b) The main producible assemblies of the 2HAM system (for clarity, not all subassemblies are shown). The target assembly is producible and terminal. However, there is also a produced assembly that is a rogue assembly (highlighted) since it not a subassembly of our target and it is terminal. (c) A small example of an assembly tree for one of the producibles.

Unique Assembly.   Intuitively, the unique assembly of A means that any produced assembly can continue to grow until it becomes A, thus making A the *uniquely produced* assembly if the process is provided sufficient time to assemble. This means A is the unique terminal assembly and all produced assemblies are subassemblies of A. Formally, we say a system Γ uniquely produces an assembly A if: 1) $P_\Gamma = \{A\}$, and 2) For all $B \in P'_\Gamma$, $|B| \leq |A|$.

▶ **Problem 1** (Unique Assembly Verification). **Input:** A 2HAM system Γ, an assembly A. **Output:** Does Γ uniquely produce the assembly A?

The Unique Assembly Verification problem (UAV) is the computational problem that asks to verify if an assembly is uniquely produced. A key concept used throughout this paper is a *rogue assembly*, which is any producible assembly that breaks one of the conditions of unique assembly and serves as a proof that the instance of the UAV problem is false.

▶ **Definition 2** (Rogue Assembly). *Given an instance of UAV (Γ, A), an assembly $R \sqsubseteq P'_\Gamma$ is a rogue assembly if $R \neq A$ and R is not a subassembly of A.*

We prove the following Lemma, which is used in the hardness reduction and the positive result. This lemma states that if the instance of UAV is false and all the tiles in Γ are used to build A, then any rogue assembly is made of combinable subassemblies of A.

▶ **Lemma 3.** *For an instance of UAV (Γ, A) that is false the following statement must be true: there exists two producible assemblies B, C such that $B, C \sqsubseteq A$ and B and C are τ-combinable into a rogue assembly R, or there exists a rogue assembly R that is composed of a single tile.*

Proof. First, since the instance of UAV is false, there must exist some rogue assembly R. If R is composed of a single tile, the Lemma is true. If R is not composed of a single tile, we walk through its assembly tree to find the assemblies B and C that are both subassemblies of our target A. Consider an assembly tree of R, Υ_R. Start at the root- if its two children are both subassemblies of A, then the rogue assembly R satisfies the Lemma. If either of the children is also a rogue assembly (not a subassembly of the target), then follow that node and do the same thing. If both are rogue assemblies, it does not matter which we follow.

Since this is an assembly tree all the leaves represent assemblies composed of one tile. Since we know none of the leaf assemblies are rogue assemblies (if it was the lemma would already be satisfied) we know at some point we must reach a node representing a rogue assembly that can be built from two subassemblies of our target A.   ◀

3   Unique Assembly Verification Hardness

In this section, we show coNP-hardness of the Unique Assembly Verification problem in the 2HAM with constant temperature by a reduction from Monotone Planar 3-SAT with Neighboring Variable Pairs.

▶ **Problem 4** (Monotone Planar 3-SAT with Neighboring Variable Pairs (MP-3SAT-NVP)).
Input: Boolean formula $\phi = C_1 \wedge \cdots \wedge C_m$ in 3-CNF form where each clause only contains positive or negated literals from $X = \{x_1, \ldots, x_n\}$. Further, any clause of ϕ with 3 variables is of the form (x_i, x_{i+1}, x_j) or $(\neg x_i \vee \neg x_{i+1} \vee \neg x_j)$, i.e., at least two of the literals are neighbors.
Output: Does there exist a satisfying assignment to ϕ?

Monotone Planar 3-SAT with Neighboring Variable Pairs was recently shown to be NP-hard in [3]. We assume the instance of the problem is a rectilinear planar embedding where each variable is represented by a unit height rectangle arranged in the *variable row*. Any planar 3SAT formula has a rectilinear encoding [19]. We also assume that every clause is a unit-height rectangle with edges connecting the clauses and the contained variables. The monotone property ensures that each clause contains either only positive or only negative literals. Thus, the clauses may be separated with all positive clauses above the variable row, and all the negative clauses below. The final restriction is neighboring variable pairs, which states that for the three variables in each clause, at least two of the variables are neighbors in the variable row. An example instance is shown in Figure 3a.

3.1   Overview

Given an instance of MP-3SAT-NVP ϕ, we build an assembly A and a 2HAM system Γ that uniquely assembles A if and only if ϕ does not have a satisfying assignment. An example instance is shown in Fig. 3a and 3c. Alternatively, Γ produces a rogue assembly if and only if there exists a satisfying assignment to ϕ.

The ability to place all positive clauses above the variables and negative clauses below, along with the neighboring variable pairs, allows the clauses to be built hierarchically from the variables up. These properties allow us to require all nested clauses be evaluated and built before the outer clause is built. Thus, we define *parent* and *child* clauses as well as *root* clauses. In Figure 3a, dotted lines connect child clauses c_1 and c_2 with their parent c_3. The root clauses are c_3 and c_5.[1]

▶ **Definition 5** (Parent/Child/Root Clause). *Given a rectilinear encoding of Monotone Planer 3-SAT, a clause C_p is a* parent *clause of* child *clause C_c, if C_p fully encloses C_c, and any other clause that encloses C_c also encloses C_p. A* root *clause is a clause without a parent.*

Since ϕ is monotone, the positive and negative clauses may be separated across the variable row. The assembly A is also separated by a horizontal bar that splits the assembly in two. This bar partially extends downward to prevent this assembly from attaching to itself. Above this bar is a subassembly that encodes the positive clauses and below the bar is a subassembly that encodes the negative clauses, which we call the positive and negative circuit, respectively.

---

[1] While a formula may have multiple clauses without a parent, the authors of [3] show that by adding additional variables, an instance may be constructed with only a single root clause. For MP-3SAT-NVP, we need at least two root clauses (one for the positive and negative sides).

(a) MP-3SAT-NVP Instance.   **(b)** Target Assembly Circuit.   **(c)** UAV Target Assembly.
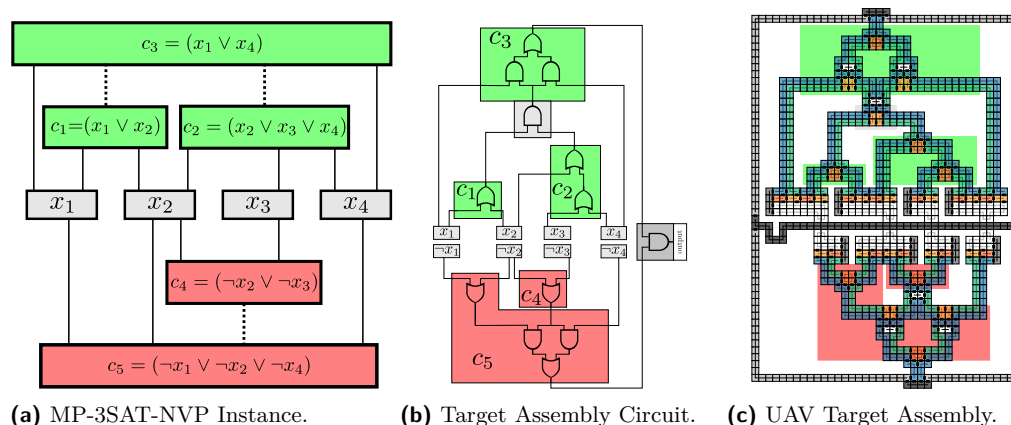
Figure 3 (a) Example instance of Monotone Rectilinear 3SAT with Neighboring Variable Pairs. Dotted lines are drawn between parent and child clauses. In this example c_3 and c_5 are the positive and negative root clauses respectively. (b) A circuit view of our example instance with gates divided into the clauses they compute. We add AND gates (shown in grey) between child clauses that have the same variables. (c) Target assembly constructed from instance on left. Each tile in the assembly is a unique tile type. Each glue is unique except for the strength 1 glues connecting the horizontal bar and the arms of each circuit. The parts of the assembly that represent each clause are boxed in.
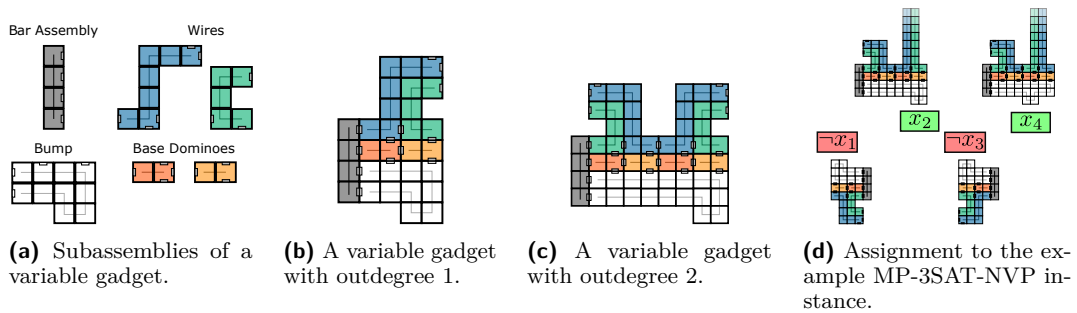
The target assembly is designed so that it must be built from the variables up to the clauses. The clause gadgets can only be built if they are satisfied. Thus, parent clauses require that their variables or child clauses be satisfied to build the gadget. We will ensure this by using AND and OR gadgets between the variable and clause gadgets. We cover the parts of the system and gadgets in the order they must assemble:

- Section 3.2: variable gadgets
- Section 3.3: OR gates and non-parent clause gadgets
- Section 3.4: AND gates and parent clauses
- Section 3.5: the root clauses and the horizontal bar
- Section 3.6: how a rogue assembly may form if and only if ϕ is satisfiable

3.2 Variable Gadget

For each variable gadget we use $(2 + 4d)$ subassemblies (Figure 4a) to build the variable gadget where d is the number of times the variable is used or its outdegree. An example is shown in Figure 4b where $d = 1$ and Figure 4c where $d = 2$. In the figures, the lines are strength-2 glues, and the rectangle glues are all strength-1, thus requiring cooperative binding for the subassemblies to attach in a specific build order. We draw our gadgets separated into subassemblies but we construct our tile set using the single tiles which will self-assemble into these subassemblies. Every variable gadget is built as follows.

- The **Bar Assembly** acts as a backbone (or separator) for the completed circuit subassemblies to connect to each other.
- The **Bump** is the first assembly to attach to the Bar Assembly. The Bump is a height 2 rectangle with an extra domino below it that is used for geometric blocking and encoding the assignment to that variable. The position of this domino is dependent on the position of the variable gadget on the opposite (negative) side.
- The **Base Dominoes** are used as part of the process of duplicating a variable path to multiple clauses. For each clause a variable is in, we use four subassemblies to connect to the next gadget. The first two gadgets are the Base dominoes. Once the Bump attaches

(a) Subassemblies of a variable gadget.

(b) A variable gadget with outdegree 1.

(c) A variable gadget with outdegree 2.

(d) Assignment to the example MP-3SAT-NVP instance.

■ **Figure 4** (a) The smaller assemblies used for building a variable gadget. (b) A variable gadget for a variable that is used in a single clause. (c) A variable gadget with outdegree 2. For each additional output more base dominoes and wires are added. (d) A set of producible subassemblies representing variables that satisfy the example instance. We will walk through how these assemblies grow into a rogue assembly.

> to the Bar Assembly, the Base Dominoes can attach cooperatively to both. Once the first Base Domino attaches the next can attach using the glue from the previous domino and the other from the Bump.[2]

▰ The **Wires** attach the variable gadgets to the clauses, and are the final two subassemblies for connecting to the next gadget. The first wire attaches cooperatively to the Bar Assembly and subsequent ones attach to the previous wire. The wires in our system are all built from two assemblies. When both halves of the wire are connected, the next gadget may attach. We call this a completed wire.

Variable gadgets in the negative circuit are built symmetrically rotated 180 degrees. We adjust the position of dominoes on the Bumps of the gadgets so that variable gadgets on opposite circuits that represent the same variable have their domino in the same column. We may generalize these gadgets to out degree d (the variable appears in d clauses) by increasing the width of the bump, and adding additional dominoes and wires.

3.3 OR Gates and Clause Gadgets

In CNF form, every variable in a clause is separated by a logical OR, thus, as part of our clause gadgets, we create OR gates to bring the variables together to ensure that the clause only forms if there is at least one variable assignment that satisfies the clause.

OR Gates. An example of how the OR gate grows off of a variable gadget is shown in Figure 5a. The OR gate consists of a single 2×2 square with strength-1 glues on the west, north, and east facing tile edges. The west and east glues each connect to the wires that input to the gate. The north glues are used cooperatively with glues on the incoming wire gadgets to attach another wire gadget going to the clause gadget. To complete the new wire gadget it must also cooperatively use the other incoming wire. Note the wires from the other input can backfill, but this does not cause an issue as the "backward" growth stops after building the wire. Figure 5b shows an example with only one variable used in the OR gate.

---

[2] Without these base dominoes, variable gadgets could build without the full Bump due to backfilling or backwards growth. The dominoes ensure this can not happen.

(a) Build OR gate.

(b) Wire backing filling.
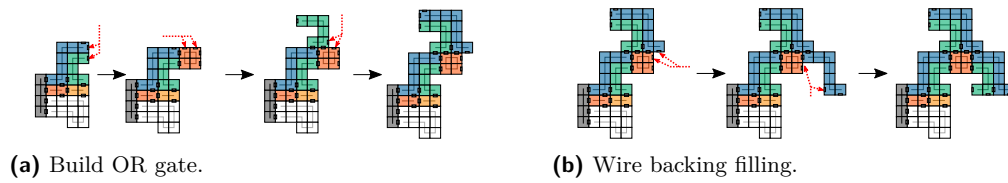
■ **Figure 5** (a) The process of a variable gadget growing the OR gate used for clauses. Glues used for attachment in the next step are denoted by arrows. If one of the variable gadgets is constructed the 2×2 square assembly may attach. The output wires of the gate then attach cooperatively with the wire from the variable assembly and the square. Note that only one of the variable assemblies needs to be constructed for the OR gate to build its output wire. (b) Once the output wires of the OR gate have attached the wire for the other variable may "Backfill" or grow backwards.

(a) Clause Gadget.

(b) Parent Clause gadget.

(c) Parent clause gadget with 3 literals.
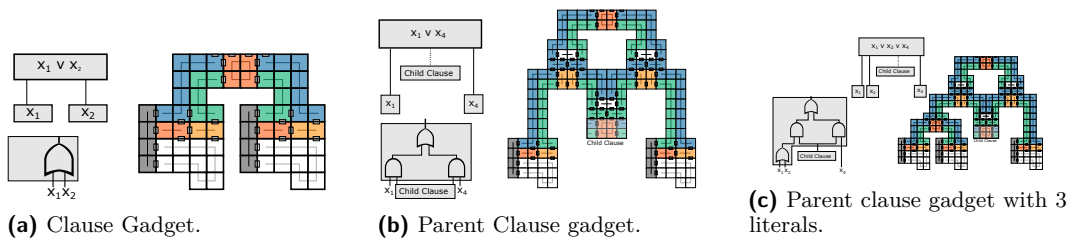
■ **Figure 6** (a) A clause gadget with 2 neighboring variables. (b) A clause gadget with two variables and a child clause. (c) When a parent clause has 3 literals we know two of them must be neighbors. Using an additional OR gate we may use the same gadget as the clause with 2 literals.

Non-parent Clauses. We first cover clauses without children, or clauses at the bottom of the circuit. The simplest type of this gadget are clauses with only 2 literals as in Figure 6a. This gadget is fairly straightforward to implement as we only need to use a single OR gate. An example of this type of clause is in Figure 7a, and its implementation is in Figure 7c. Note that both variables appear in other clauses so those variable gadgets have additional wires. For non-parent clauses with 3 literals (Figure 10a), we use 2 OR gates (Figure 8c).

3.4 AND Gates and Parent Clauses

Since every clause in CNF form is separated by a logical AND, we create AND gates that compare clauses. Thus, we need to know which clauses are parent clauses since they have child clauses underneath them with wires coming into the gates. We also build a FANOUT gate for connecting clauses.

AND Gates. The AND gate uses 2 vertical dominoes that share a single strength-1 glue between them. Figure 9a shows an example AND gate being constructed. Once a wire that inputs to the gate is completed, one of the dominoes can cooperatively attach. The domino has another strength-1 glue on its north side that allows a horizontal domino to cooperatively attach using the glue exposed on the wire.

Using the glues from the newly attached dominoes, the two halves of the gate are able to attach to each other. This allows for the two glues on the horizontal dominoes to be used to cooperatively bind the white center domino. From here, the two halves of the wire that outputs from the AND gate can attach.

(a) Clause c_1. **(b)** Circuit for c_1. **(c)** Gadgets for c_1.
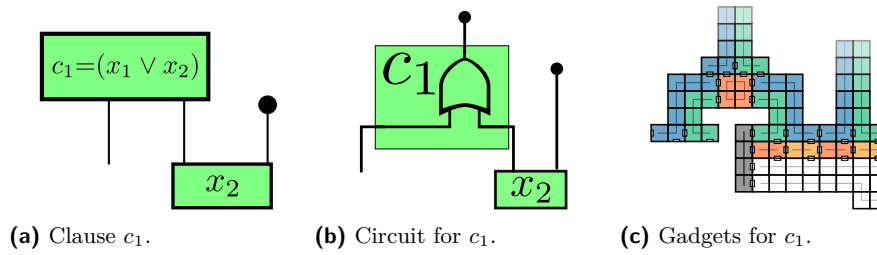
■ **Figure 7** (a) The clause c_1 in Figure 3a is satisfied by $x_2 = 1$. (b) The OR gate grows off of x_2. The other wires on the variable gadget are used to connect to other clause gadgets. (c) The gadget constructed for the clause c_1. Note the other wire from the OR gate has backfilled.

(a) Clause c_2. **(b)** Circuit for c_2. **(c)** Gadgets c_3.
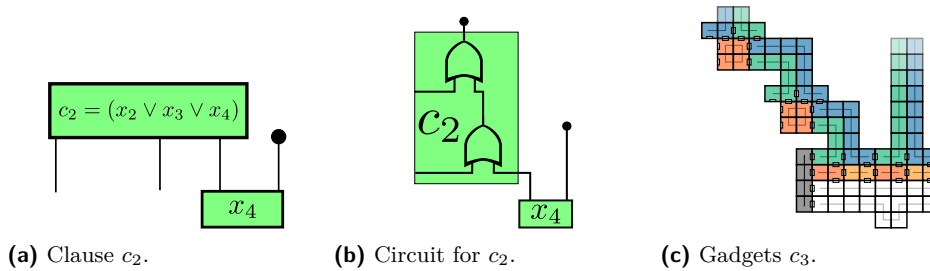
■ **Figure 8** (a) c_2 from our example. This clause has 3 variables and no children. (b) The clause is computed using two OR gates. The gates are able to grow from x_4. (c) x_4 variable gadget allows for the two OR gates to attach.

FANOUT Gates. In order to build the parent clause, we also need a way to "fan-out" and copy the signal from an AND gate to two other gadgets. We do this by adding glues to the north side of the center domino and having two wires grow off of the gadget. This process is shown in Figure 9b.

Parent Clauses. Consider a parent clause $C_p = (x_1 \lor x_4)$. Let C_c be the child clause. Since we want this gadget to build only if its own clause and its child are both satisfied, we can view this statement as $(x_1 \lor x_4) \land C_c$. However, we can modify the statement to be $(x_1 \land C_c) \lor (C_c \land X_4)$, which we can build since we have planar circuits. An example of the circuit and gadgets are shown in Figure 6b.

By the neighboring variable pairs restriction, we know that any clause with three variables has at least a pair of them being neighbors. This means that there cannot be any child clauses beneath that neighboring pair, so we may use an OR gate between those two variables and then build the rest of the gadget in the same way as the two literal version (Figure 6c).
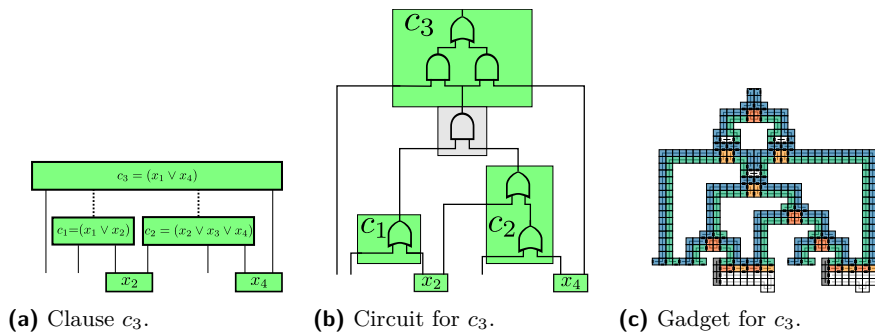
In our example instance, the root clause of the positive circuit has two children. For these cases we may use the AND gadget to verify that both child clauses have been satisfied before allowing the parent clause to build. The root clause of the negative circuit in our example instance (Figure 11a) has three literals. The constructed gadget can be seen in Figure 11a.

3.5 Root Clauses and Horizontal Bar

Root Clauses and Arms. The root clause is the outermost clause on either side of the variables. Although it functions similar to the other clauses, instead of outputting a wire, a horizontal 4×1 rectangle can attach after it finishes assembling. The arms may then cooperatively bind to the rectangle and the wires of the root clause forming the top of the

(a) Build AND gate.

(b) Build FANOUT.

(c) Attachment of an AND gate.
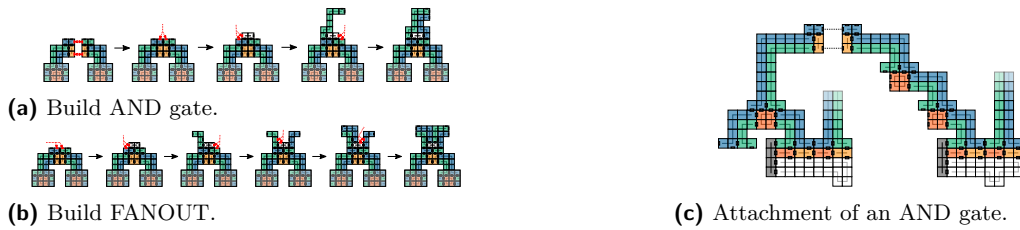
Figure 9 (a) The process of an AND gadget assembling. The output wires can only grow from the combined halves of the AND gate. (b) By modifying the center domino, two wires may be output from a single AND gate, which works as a FANOUT. (c) The two clauses c_1 and c_2 both have the same parent clause so they are joined by an AND gate. Once the dominoes attach to the output wire of the clauses the two assemblies may attach to each other.

(a) Clause c_3.

(b) Circuit for c_3.

(c) Gadget for c_3.

Figure 10 (a) The root clause of the example instance. This clause has two literals and two children. (b) Since the AND gate has built and x_4 satisfies c_3 the clause may grow. (c) The two child clause's output are connected by an AND gate and then used as the middle input to the gadget.

circuit. The glues on the ends of these arms allow for the circuit to attach to the horizontal bar. A high-level view of the root clauses and arms attached is shown in Figure 12a as well as a detail of the assembly process of the root clause in Figure 12b.

Horizontal Bar. The horizontal bar (Figure 12a) is a width-1 assembly that extends the width of both circuits with strength-1 glues on the north and south side of the outer tiles. Since the arms must also be able to attach to each other to form a rogue assembly the glues on the ends of the horizontal bar must be the same. In order to prevent the horizontal bar from attaching to another instance of itself, we extend the bar partially downward so it will geometrically block copies from attaching.

3.6 Rogue Assemblies

For the construction of the target assembly, each piece is built from the variables up to the root clause. However, the nondeterministic build order means that not all parts of each circuit need to be built in order for the root clause to be satisfied. For instance, if one of the variables in a clause attaches, the OR gates will still allow the wires to attach. Thus, using a variable constitutes setting it to true (and in the negative circuit using a variable is setting the negation to true).

With root clauses satisfied and the arms attaching, a rogue assembly may occur as shown in Figure 13c. The corresponding circuit is shown in Figure 13b. This can occur because the arms can attach to each other without the horizontal bar. Normally, the variable gadgets

(a) Clause c_5. **(b)** Circuit for c_5. **(c)** Gadget for c_5.
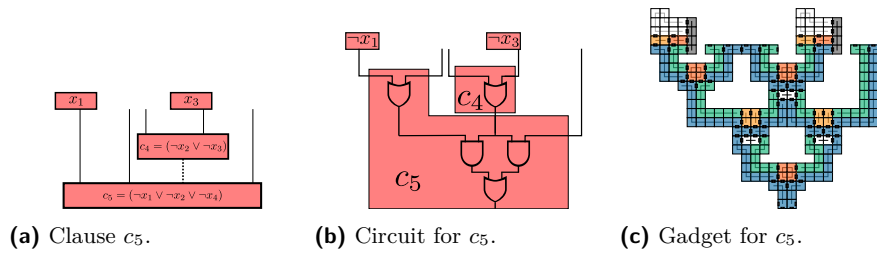
Figure 11 (a) In the example instance the negative circuit has c_5 which is a parent clause with 3 literals. (b) The negative circuit draw with gates. The variables x_1 and x_3 being false satisfies all the clauses. (c) The variable assemblies we selected at the beginning also grow into a circuit with the root clause built.

(a) Horizontal Bar and Completed Circuits. **(b)** Glues on root clause and variable gadgets.
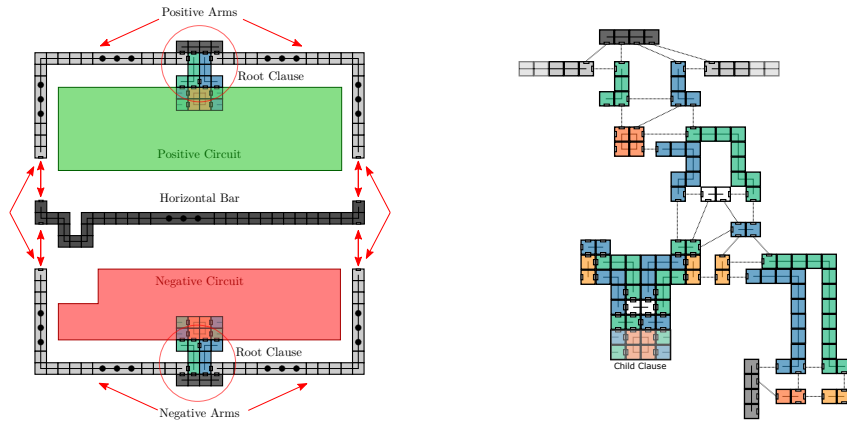
Figure 12 (a) The root clauses and arms joining the positive and negative assemblies with the horizontal bar. The root clause allows a short wire to attach where the arms can then attach. Each arm has a strength-1 glue at the end. The horizontal bar separates the positive circuit from the negative circuit. The small bump is so the bar can not attach to other horizontal bars. (b) Each subassembly of the root clause cannot attach to each other without being satisfied from the child clauses since each subassembly only shares a strength-1 glue with adjacent assemblies.

would overlap and prevent this attachment if both the positive and the negative circuit used the same variable (which is setting a variable to both true and false). Thus, the positive and negative side each have their own set of variables that make all clauses on their respective sides true. This rogue assembly can only happen if there is a subset for each side that allows all clauses to be true, and thus satisfies the original MP-3SAT-NVP formula.

For an MP-3SAT-NVP instance ϕ and an assignment X_s to the variables in X, let A_p and A_n be the positive and negative circuit assemblies, respectively, created from ϕ. We say an assembly $A_p' \sqsubseteq A_p$ represents the assignment X_s if it has attached variable gadgets for the variables in X_s that equal 1, and has built its root clause. For negated circuits, it must have variable gadgets attached for variables set to 0 in X_s.

▶ **Lemma 6.** *For a rectilinear encoding of Monotone Planar 3SAT ϕ with neighboring variable pairs and 2HAM system Γ_ϕ as described above, there exist two producible assemblies $A_p' \sqsubseteq A_p$ and $A_n' \sqsubseteq A_n$ that both represent the same assignment X_s to the variables X, if and only if X_s satisfies ϕ.*

Proof. If there exists a satisfying assignment X_s to X, we may build A'_p by taking the variable gadgets for variables assigned to 1 and grow the circuit off of them. Since we know all the clauses are satisfied, each clause gadget (including the root clause) may grow resulting in an assembly A'_p that represents X_s. By the same argument we know A'_n is producible since X_s satisfies ϕ, which includes the negated clauses.

We prove these assemblies are producible only if X_s satisfies ϕ via contradiction. Assume X_s does not satisfy ϕ, but both assemblies A'_p and A'_n are producible. Since X_s does not satisfy ϕ, there must exist at least one unsatisfied clause c_i. W.L.O.G., assume c_i is a positive clause. We show the assembly A'_p cannot be produced.

If c_i is the root clause, assume all of the children of c_i are satisfied. The center input of the clause is a producible subassembly of A'_p since variable gadgets are allowed that satisfy the clauses below it. We can see in Figure 12b the other producible subassemblies of the gadgets only have a strength-1 glue between them. This means none of the subassemblies are able to attach to each other on their own. In order for the arms to attach to the output wire of the root clause, at least one of the AND gates must be fully constructed. The AND gate cannot assemble unless both halves of the gate have been constructed. The middle input is built, but the other half of the AND gadget must grow off a completed wire from the variable gadget. However, since c_i is not satisfied the variables gadgets which satisfy the formula have not attached so the assembly A'_p cannot build the clause gadget.

If c_i is another parent clause that is not the root. Let the clause c_j be the parent clause of c_i. If the clause gadget for c_i is not constructed then the gadget for c_j is not buildable. Since the gadgets used are the same as the root clause, the output wires of the clause gadget for c_i cannot be built without a variable gadget which satisfies the formula. The middle input of clause gadget representing c_j will not be buildable since this would be the output wire of c_i. The middle input goes to two AND gates that cannot construct unless both wires have been built. Thus, the output wire of c_j cannot be built without its children clauses satisfying it. In the case c_j has multiple children, the output wires of all its children are joined by AND gates that will not construct without both inputs.

Finally, consider the case where c_i is a clause without children. In order for the clause's output wire to complete, it must be attached to an OR gadget and the outer wire of the variable gadget. The OR gadget may only attach to a completed wire from a variable gadget (or another OR). The variable gadget cannot be completed without placing the bump, so we cannot have built the outwire of c_i. By the same argument as the previous case, this clause not being built results in its parent not being built. If c_i is not satisfied, the clause gadget for c_i cannot be constructed, which means the assembly A'_p is not producible. ◀

▶ **Theorem 7.** *The Unique Assembly Verification problem in the 2HAM is coNP-Complete with $\tau = 2$.*

Proof. Given an instance of a rectilinear encoding of Monotone Planar 3SAT with neighboring variable pairs ϕ, we provide an explicit polynomial time reduction by creating a 2HAM system $\Gamma = (T, 2)$ and an assembly A such that Γ uniquely produces A if and only if there does not exist a satisfying assignment to ϕ. We create the assembly A by taking the rectilinear encoding of ϕ, arranging the rectangles on a grid graph, and replacing the rectangles with the given variable and clause gadgets. We also add the arms and horizontal bar.

Assume there exists a satisfying assignment X_s to the variables X, for ϕ. We know by Lemma 6, there exist two producible assemblies A'_p and A'_n that both contain the arms and have complementary bump positions[3]. These two assemblies can cooperatively bind to one another using the two glues on their arms, and thus produce a rogue assembly as in Figure 13c. This means a satisfying assignment to ϕ implies Γ does not uniquely construct A.

---

[3] Having complimentary bump positions is equivalent to both representing the same assignment.

(a) Satisfying Assignment.   **(b)** Rogue Assembly Circuit.   **(c)** Rogue Assembly.
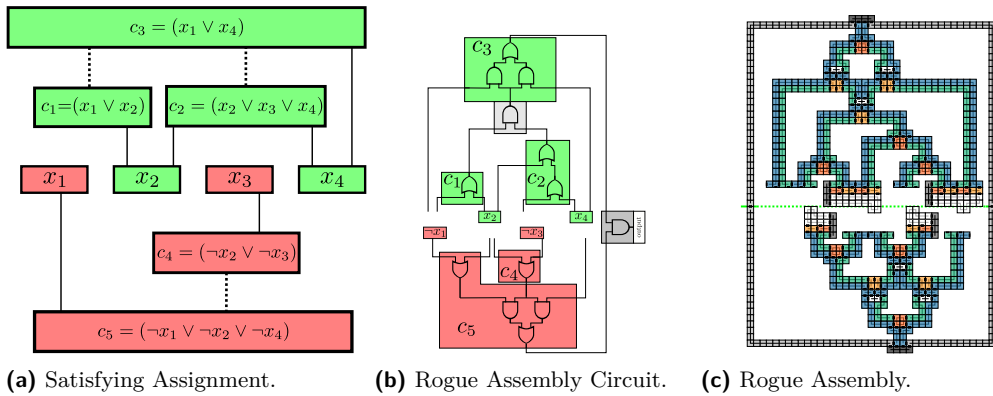
Figure 13 (a) There exists a satisfying assignment for the example instance with green blocks representing variables which equal 1 and red blocks representing 0. (b) Rogue assembly drawn as a circuit with selected variables. (c) The 2HAM system will produce a Rogue Assembly from the two circuit assemblies which represent the satisfying assignment.

(a) Tree-bonded and Shaped.   **(b)** Tree-bonded.   **(c)** Not Tree-bonded.
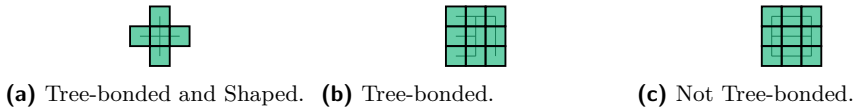
Figure 14 (a) Both the shape and bond graph of this assembly are trees. (b) Even though the shape of this assembly is a square, its bond graph is acyclic, and thus this assembly is tree-bonded. (c) This assembly is not tree-bonded due to the cycle in its bond graph.

Now assume Γ does not uniquely produce A, so there exists some rogue assembly B. The only repeated glues in the tile set of Γ are the exposed glues on the arms. Any rogue assembly must use these two glues to assemble, and they must be assembled from two subassemblies of the target by Lemma 3. Let B be producible by combining two assemblies b and b'. Since both b and b' are producible assemblies with both their arms, and they can attach to each other, they are not geometrically blocked. This implies they must represent the same assignment and by Lemma 6, this can only be true if the assignment satisfies ϕ. By viewing which variable gadgets are included in the two assemblies, we can identify the satisfying assignment to ϕ. Thus, Γ will uniquely produce A if and only if there does not exist a satisfying assignment to ϕ.   ◀

4   Verification of Tree-Bonded Assemblies

In this section, we investigate the problem of Unique Assembly Verification with the promise that the target assembly A is *tree-bonded*, meaning the bond graph of the target assembly forms a tree. Figures 14a and 14b show examples of tree-bonded assemblies. Figure 14c shows an assembly whose bond graph contains a cycle and thus is not a tree-bonded assembly. We first present a $\mathcal{O}(|A|^5)$ algorithm for temperature 2 systems, and then extend this method to provide a $\mathcal{O}(|A|^5 \log \tau)$ time dynamic programming algorithm for the case where the temperature τ of the system can be passed as a parameter. Before describing the algorithms, we first introduce some required definitions and the problem formulations.

Tree-Bonded Assemblies.   An assembly A is a *tree-bonded assembly* if and only if the induced bond graph $\mathcal{G}_A$ is acyclic.

Binding Sites. For two configurations C_1 and C_2, we say a *binding site* $\mathcal{B}$ is a pair of points (p_a, p_b), such that their distance is $||p_a - p_b||_2 = 1$, and the tiles $C_1(p_a)$ and $C_2(p_b)$ have nonzero glue strength between each other. The set of binding sites for two configurations is the set of pairs of points that meet this requirement. We also define an *inner binding site*. For two configurations, C_1 and C_2, and a pair of binding sites $a = (a_1, a_2'), b = (b_1, b_2')$, let $I(a, b)$ be the set of binding sites that occur on the inside of the loop formed by a, b (inner binding sites). An example of the area enclosed by a loop is seen in Figure 15c.

Simple Loops. For two configurations, C_1 and C_2, and a pair of binding sites $a = (a_1, a_2'), b = (b_1, b_2')$, we say the loop formed by a, b is a *simple loop* if $|I(a, b)| = 0$.

Origin Configuration In discussing different configurations and assemblies, it is useful to anchor a configuration to a fixed point. For an assembly A, the *origin configuration* A_0 is the translation of $A's$ configuration such that the bottom left vertex of the bounding box of elements in $dom(A_0)$ is at the origin $(0, 0)$.

▶ **Problem 8** (TREE-UAV). **Input:** A 2HAM system Γ and a tree-bonded assembly A. **Output:** Does Γ uniquely produce the assembly A?
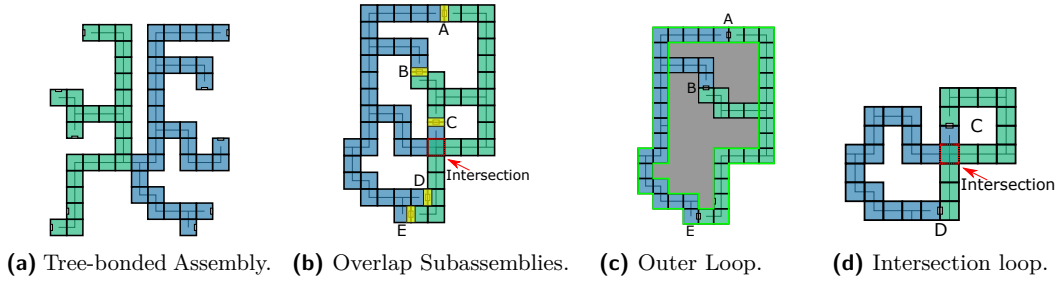
4.1 High-level Overview

The high-level goal of this algorithm is to find a rogue assembly that acts as a witness that the instance of UAV is false. We note that a given instance, $P = (\Gamma, A)$, of TEMP2-TREE-UAV, where $\Gamma = (T, 2)$, can be broken down into three possible cases. An example tree-bonded assembly is shown in Figure 15a.

1. The instance P is false, and Γ produces a tree-bonded rogue assembly.
2. The instance P is false, and the only rogue assemblies producible in Γ are non tree-bonded.
3. The instance P is true.

At a high level, the algorithm checks if either Case 1 or Case 2 as true, and f so, the algorithm rejects, otherwise it accepts. Case 1 can be checked efficiently by modifying Γ to function as a noncooperative system Γ' and utilizing the algorithm for temperature-1 UAV provided in [16]. To check the second case, Lemma 3 states that if the instance is false, it suffices to check pairs of subassemblies of the target assembly A in order to find a witness rogue assembly. Thus, we take two copies of the target assembly and attempt to find possible ways they may bond, even if the resulting assembly places two tiles at the same position. We call the pairs of tiles that contribute glue strength *binding sites* Tiles that are in the same position are called *intersections*. An example of both may be seen in Figure 15b.

We first analyze the case of temperature-2 systems where only two binding sites that do not intersect are needed. We then generalize this algorithm by using dynamic programming to find the set of binding sites to maximize the binding strength between the assemblies without any intersections.

▶ **Problem 9** (TEMP2-TREE-UAV). **Input:** A $\tau = 2$ 2HAM system Γ and a tree-bonded assembly A. **Output:** Does Γ unique produce the assembly A?

(a) Tree-bonded Assembly.   **(b)** Overlap Subassemblies.   **(c)** Outer Loop.   **(d)** Intersection loop.

Figure 15 (a) An example tree-bonded target assembly. (b) One possible overlap configuration formed by two subassemblies with 5 binding sites that are highlighted. (c) The loop formed by binding sites A and E is outlined in green. Any binding site that occurs in the grey shaded area, such as B, is in the set of inner binding sites for (A, E). (d) The loop formed using binding sites (C, D) intersects itself and cannot be used.

Algorithm 1 NonCoop-UAV(Γ, A). The runtime of Temp1-UAV is $\mathcal{O}(|A||T| \log |T|)$ [16].

Data: 2HAM System $\Gamma = (T, \tau)$, an assembly A
Result: Does Γ uniquely assembly A if it can only utilize strength $\geq \tau$ glues?
Modify T to create T' by removing all glues of strength less than τ, and setting the
  strength of all glues of strength $\geq \tau$ to 1;
if *Temp1-UAV*$(\Gamma' = (T', \tau = 1), A)$ **then**   accept;
else reject;

4.2   Tree-Bonded Rogue Assemblies

The following algorithm checks if a system uniquely assembles a given shape provided the system is restricted to behaving in a noncooperative manner. This means that two assemblies can only attach if they share one or more strength-τ glues between them. This system functions equivalently to a temperature-1 system where all glues less than strength-τ are removed and all glues greater than strength-τ are set to strength-1. We modify the system in this way and then use the known polynomial time algorithm for temperature-1 UAV [16].

▶ **Lemma 10.** *For any 2HAM system $\Gamma = (\Sigma, \tau)$ and tree-bonded assembly A, if NonCoop-UAV(Γ, A) (Algorithm 1) is true, and Γ does not uniquely assemble A, then there exists assemblies B, B_1, B_2, s.t. $B \not\sqsubseteq A$, $B_1, B_2 \sqsubseteq A$, B_1 and B_2 combine to form B by utilizing cooperative binding.*

Proof. Since NonCoop-UAV(Γ, A) is true, but Γ does not uniquely assemble A, there must exist a rogue assembly $B' \not\sqsubseteq A$ since any subassembly of A would be tree-bonded. Consider an assembly tree Υ'_B for B'. Since NonCoop-UAV(Γ, A) is true, the singleton tile leaves of Υ'_B must be subassemblies of A. We will show there exists a node $B \in \Upsilon'_B$, with children B_1 and B_2, respectively, such that $B' \not\sqsubseteq A$, and $B_1, B_2 \sqsubseteq A$.

Let the root node of the tree Υ_B be the candidate node B, and let assemblies B_1 and B_2 be the two children of B. If B_1 and B_2 are both subassemblies of A, then the conditions are met. Otherwise, W.L.O.G. assume $B_1 \not\sqsubseteq A$. Now set the candidate node B to B_1 and repeat. Since all leaves of Υ_B represent subassemblies of A, there must be a point in which the candidate B node is some assembly $B \not\sqsubseteq A$, and its children are assemblies $B_1, B_2 \sqsubseteq A$. ◀

▪ **Algorithm 2** Algorithm to solve TEMP2-TREE-UAV in $\mathcal{O}(|A|^5)$ time.

Data: 2HAM System $\Gamma = (\Sigma, 2)$, Tree-Bonded Assembly A of height h and width w
Result: Does Γ uniquely produce A?
if *NONCOOP-UAV(Γ, A) rejects* **then** reject;
Let A_0 be the origin configuration of assembly A;
for $i \leftarrow -w$ **to** w **do**
    for $j \leftarrow -h$ **to** h **do**
        $A' \leftarrow A_0 + \langle i, j \rangle$;
        Let $\mathcal{B}$ be the set of binding sites between A_0 and A';
        for *each pair of binding sites $a, b \in \mathcal{B}$* **do**
            if *The loop formed using a, b does not intersect itself* **then** reject;
accept;

4.3 Temperature-2

With respect to the given instance of TEMP2-TREE-UAV P, if P is false, and the algorithm for NONCOOP-UAV(Γ, A) returns "accept", then Lemma 10 implies there exist two subassemblies of the target, B_1 and B_2, that attach to each other using cooperative binding.

To find these two subassemblies, we take two "copies" of the target assembly and find all $|A|^2$ possible ways to combine them- even if it results in intersections. If any combination results in at least two binding sites, we attempt to find 2-combinable subassemblies. Since these subassemblies are also tree-bonded, there only exists one path between each pair of tiles- including the binding sites. So for each pair of binding sites, we take the loop formed by the two binding sites and check if it intersects itself. An example of a loop that intersects itself is shown in Figure 15d. If there ever exists a pair of binding sites whose paths do not intersect, then those two subassemblies will form a rogue assembly and we reject.

▶ **Theorem 11.** *There is a $\mathcal{O}(|A|^5)$ time algorithm that decides TEMP2-TREE-UAV.*

Due to space constraints, the analysis of Alg. 2 and proof of Thm. 11 have been omitted.

4.4 Variable Temperature

We now present an algorithm for TREE-UAV as a generalization of the previous problem where the temperature of the system τ is given as input. The algorithm is similar except it does not suffice to only find a single loop since the temperature requirement attachment may be greater than 2. We must find multiple loops between binding sites that do not intersect. Once we find a way to combine the assemblies, we view binding sites and loops hierarchically using inner binding sites. An example of an inner binding site is in Figure 15c. We recursively calculate the max binding strength when taking each pair of binding sites as the outer loop. After calling NONCOOP-UAV(Γ, A), we check each possible way to attach A to itself. For each of these ways, we build a $b \times b$ table where b is the total number of induced binding sites. For each pair of binding sites, we calculate the maximum value recursively augmented with the table. Thus, we only compute the maximum value once for each loop.

First, if the created loop intersects itself, we cannot use it, so the value is set to -1. Next, we check if the binding sites form a simple loop with no inner binding sites. Here, the max value is the sum of the glue strengths between the binding sites. For loops with inner binding sites, we do a *loop decomposition*, which is the process of breaking a loop into two smaller loops along one of the inner binding sites. An example is shown in Figure 16a.

(a) Loop Decomposition.   **(b)** Invalid Loop Decomposition.
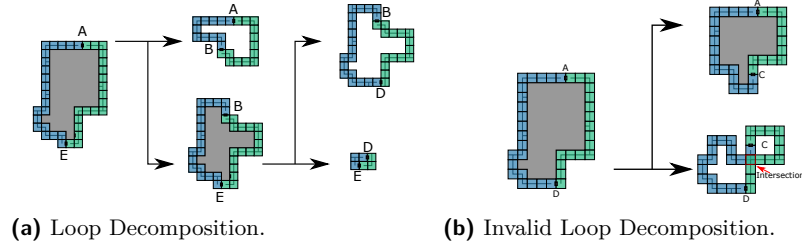
Figure 16 (a) One possible way to decompose loops into simple loops based on inner binding sites. (b) Decomposing the loop (A, D) along binding site C results in the loop (C, D) which intersects itself. This means we cannot decompose the loop (A, D) along C.

Algorithm 3 Algorithm to solve TREE-UAV in $\mathcal{O}(|A|^5 \log \tau)$ time.

Data: 2HAM System $\Gamma = (\Sigma, \tau)$, Tree-Bonded Assembly A of height h and width w
Result: Does Γ uniquely produce A?
if *NONCOOP-UAV*(Γ, A) *rejects* **then** reject;
Let A_0 be the origin configuration of assembly A;
for $x \leftarrow -w$ **to** w **do**
     for $y \leftarrow -h$ **to** h **do**
         $A' \leftarrow A_0 + \langle x, y \rangle$;
         Let $\mathcal{B}$ be the set of binding sites between A_0 and A' and let $b = |\mathcal{B}|$;
         Create a $b \times b$ table $T_{\mathcal{B}}$ indexed by the elements of $\mathcal{B}$ with all cells empty;
         for *each pair of binding sites* $b_1, b_2 \in \mathcal{B}$ **do**
             if $maxStr(C = (A_0 \bigcup A'), T_{\mathcal{B}}, b_1, b_2) \geq \tau$ **then** reject;
accept;

To find the max binding strength of the outer loop, we break the loop up along each inner binding site and recursively get the max strength of the two resulting loops (subtracted by the glue strength of the inner binding site since it would be counted twice). If either of the smaller loops intersects itself, it returns -1 and we know not to use that inner binding site. The max binding strength of the outer loop is then the maximum of these computed values over all choices of inner binding sites. The recursive checks are implemented with a dynamic programming/memoization table to eliminate repeated recursive calls.

▶ **Theorem 12.** *There is a $\mathcal{O}(|A|^5 \log \tau)$ time algorithm that decides* TREE-*UAV.*

Due to space constraints, the analysis of Alg. 3 and proof of Thm. 12 have been omitted.

5   Conclusion

In this paper, we have addressed the long-standing open problem of the complexity of verifying unique assembly within the 2-handed tile self-assembly model and shown that the problem is coNP-complete even at temperature $\tau = 2$ and in two dimensions. These are the smallest possible values for which this problem can be hard, as both temperature-1 self-assembly and 1-dimensional self-assembly have established polynomial time verification solutions. Given this hardness, we explored a natural scenario where this problem might be more tractable, and showed that restricting the input assemblies to tree-bonded assemblies allows for an efficient $\mathcal{O}(|A|^5 \log \tau)$-time unique assembly verification algorithm.

■ **Algorithm 4** $maxStr(C, T_{\mathcal{B}}, b_1, b_2)$. The subroutine that calculates the max strength when using two binding sites as the outer loop. The method $glueStr(b)$ takes in a binding site and returns the strength of the glue connecting the two tiles.

Data: Union of two assemblies C, Table T_B, and Binding sites b_1, b_2
Result: The maximum binding strength used to build a stable subassembly of C.
if $T_{\mathcal{B}}(b_1, b_2)$ *is empty* **then**
    if *The loop formed by* b_1, b_2 *intersects itself* **then** return -1;
    if $|I(b_1, b_2)| = 0$ **then** return $glueStr(b_1) + glueStr(b_2)$;
    Let $T_B(b_1, b_2) = 0$;
    for $b_i \in I(b_1, b_2)$ **do**
        if $maxStr(b_1, b_i) \neq -1$ *AND* $maxStr(b_i, b_2) \neq -1$) **then**
            $s \leftarrow maxStr(b_1, b_i) + maxStr(b_i, b_2) - glueStr(b_i)$;
            if $s > T_{\mathcal{B}}(b_1, b_2)$ **then** $T_{\mathcal{B}}(b_1, b_2) \leftarrow s$;
return $T_{\mathcal{B}}(b_1, b_2)$;

Future Work.   While we have resolved the general question of unique assembly verification in the 2HAM, as well as addressed a natural restricted case of tree-bonded assemblies, there remain important directions for future research.

- Our hardness reduction utilizes a tile set that is roughly the size of the input assembly. All hardness results in the literature for the 2-handed self-assembly model have this property. Yet, the computational power of self-assembly allows for the assembly of large assemblies with small tile sets, as seen in the efficient self-assembly of squares [1, 24], or the implementation of "Busy Beaver" Turing machines [24]. How hard is UAV for large assemblies with substantially smaller tile sets. Does the hardness scale with assembly size or tile set size? Is there some form of fixed-parameter tractability for the UAV problem?

- We proved that UAV for the multiple tile (or q-tile) model is coNP-complete with polynomial-sized assemblies attaching. Is UAV polynomial in the multiple tile model and the 2HAM in the case where every producible assembly, except the one that grows into the terminal assembly, is bounded by a constant?

- A related question in the aTAM and the 2HAM is the number of two-handed operations actually required to make the problem hard. If all subassemblies can only grow by single tile attachments, how many two-handed operations to combine those subassemblies are needed for UAV to remain hard? The ability to more efficiently construct shapes by assembling parts separately has been studied in other models as well [25].

- Another direction initiated by our efficient tree assembly algorithm is the consideration of other natural restricted classes of the UAV problem. For example, how does UAV scale with respect to the genus of an assembly's connectivity graph? A related question involves verification for *fully connected* assemblies, a previously-studied concept [14] in which assemblies include positive bonds between all neighboring tiles.

—— **References** ——

1   Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, STOC'01, pages 740–748, 2001. `doi:10.1145/380752.380881`.

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 *Proc. of the 34th ACM Sym. on Theory of Computing*, pages 23–32, 2002.

3   Pankaj K. Agarwal, Boris Aronov, Tzvika Geft, and Dan Halperin. On two-handed planar assembly partitioning with connectivity constraints. In *Proc. of the ACM-SIAM Syme on Discrete Algorithms*, SODA'21, pages 1740–1756, 2021. `doi:10.1137/1.9781611976465.105`.

4   Gagan Aggarwal, Qi Cheng, Michael H. Goldwasser, Ming-Yang Kao, Pablo Moisset de Espanes, and Robert T. Schweller. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34(6):1493–1515, 2005. `doi:10.1137/S0097539704445202`.

5   David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. The complexity of multiple handed self-assembly. In *Proceedings of the International Conference on Unconventional Computation and Natural Computation*, UCNC'21, pages 1–18, 2021.

6   David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. Covert computation in staged self-assembly: Verification is pspace-complete. In *Proceedings of the 29th European Symposium on Algorithms*, ESA'21, pages 23:1–23:18, 2021.

7   David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. Complexity of verification in self-assembly with prebuilt assemblies. In *Proceedings of the Symposium on Algorithmic Foundations of Dynamic Networks*, SAND'22, 2022.

8   Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, David Furcy, Matthew J. Patitz, Robert Schweller, Scott M. Summers, and Andrew Winslow. On the effects of hierarchical self-assembly for reducing program-size complexity. *Theoretical Computer Science*, 894:50–78, 2021. `doi:10.1016/j.tcs.2021.09.011`.

9   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 Inter. Sym. on Theoretical Aspects of Computer Science*, volume 20 of *STACS'13*, pages 172–184, 2013.

10   Angel A. Cantu, Austin Luchsinger, Robert Schweller, and Tim Wylie. Covert Computation in Self-Assembled Circuits. *Algorithmica*, 83:531–552, 2021. arXiv:1908.06068.

11   Cameron Chalk, Dominic Fernandez, Alejandro Huerta, Mario Maldonado, Robert Schweller, and Leslie Sweet. Strict self-assembly of fractals using multiple hands. *Algorithmica*, 76, July 2014. `doi:10.1007/s00453-015-0022-x`.

12   Ho-Lin Chen and David Doty. Parallelism and time in hierarchical self-assembly. *SIAM Journal on Computing*, 46(2):661–709, 2017. Preliminary version appeared in SODA 2012.

13   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.

14   Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Comp. Science*, 671:4–18, 2017.

15   David Doty. Theory of algorithmic self-assembly. *Comm. of the ACM*, 55(12):78–88, 2012.

16   David Doty. Producibility in hierarchical self-assembly. In *Proc. of the Inter. Conf. on Unconventional Computation and Natural Computation*, UCNC'14, pages 142–154, 2014.

17   Algorithmic Self-Assembly Research Group. VersaTile. github.com/asarg/VersaTile, 2014.

18   Jacob Hendricks and Joseph Opseth. Self-assembly of 4-sided fractals in the two-handed tile assembly model. *Natural Computing*, 18:75–92, 2018.

19   Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM Journal on Discrete Mathematics*, 5(3):422–427, 1992. `doi:10.1137/0405033`.

20   Pierre-'Etienne Meunier, Damien Regnault, and Damien Woods. The program-size complexity of self-assembled paths. In *Proc. of the 52nd Annual ACM SIGACT Sym. on Theory of Computing*, STOC 2020, pages 727–737, 2020. `doi:10.1145/3357713.3384263`.

21   Pierre-'Etienne Meunier and Damien Woods. The non-cooperative tile assembly model is not intrinsically universal or capable of bounded turing machine simulation. In *Proc. of the 49th Annual ACM SIGACT Sym. on Theory of Computing*, STOC'17, pages 328–341, 2017.

22   Matt Patitz. Pytas. URL: `http://self-assembly.net/wiki/index.php?title=PyTAS`.

23   Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, June 2014.

24   Paul WK Rothemund and Erik Winfree. The program-size complexity of self-assembled squares. In *Proc. of the 32nd annual ACM Sym. on Theory of computing*, pages 459–468, 2000.

25   Arne Schmidt, Sheryl Manzoor, Li Huang, Aaron Becker, and Sándor P. Fekete. Efficient parallel self-assembly under uniform control inputs. *Robotics and Automation Letters*, 3:3521–3528, 2018.

26   Robert Schweller, Andrew Winslow, and Tim Wylie. Complexities for high-temperature two-handed tile self-assembly. In *Proc. of the 23rd Inter. Conf. on DNA Computing and Molecular Programming*, DNA'17, pages 98–109, 2017.

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

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

29   Erik Winfree, Rebecca Schulman, and Constantine Evans. The xgrow simulator. URL: `https://www.dna.caltech.edu/Xgrow/`.

30   Damien Woods. Intrinsic universality and the computational power of self-assembly. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 373(2046):20140214, 2015.

31   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(7748):366–372, 2019. `doi:10.1038/s41586-019-1014-9`.