Complexity of Verification in Self-Assembly with Prebuilt Assemblies

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

We analyze the complexity of two fundamental verification problems within a generalization of the two-handed tile self-assembly model (2HAM) where initial system assemblies are not restricted to be singleton tiles, but may be larger pre-built assemblies. Within this model we consider the *producibility* problem, which asks if a given tile system builds, or produces, a given assembly, and the *unique assembly verification* (UAV) problem, which asks if a given system *uniquely* produces a given assembly. We show that producibility is NP-complete and UAV is coNP^{NP}-complete even when the initial assembly size and temperature threshold are both bounded by a constant. This is in stark contrast to results in the standard model with singleton input tiles where producibility is in P and UAV is in coNP for $\mathcal{O}(1)$ bounded temperature and coNP-complete when temperature is part of the input. We further provide preliminary results for producibility and UAV in the case of 1-dimensional *linear* assemblies with pre-built assemblies, and provide polynomial time solutions.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Applied computing → Computational biology

Keywords and phrases 2-handed assembly, verification, prebuilt

Digital Object Identifier 10.4230/LIPIcs.SAND.2022.8

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

1 Introduction

Self-Assembly is the process by which a system of simple particles autonomously come together to form complex structures. *Algorithmic* self-assembly studies scenarios in which dynamics of system molecules encode computation, allowing for algorithmic control of the self-assembly of matter. A premiere model for the study of algorithmic self-assembly is the *tile self-assembly model* [3, 17], in which system monomers are modeled as four-sided Wang tiles that randomly collide and combine based on matching tile edges and a given bonding threshold called the *temperature*. Tile self-assembly has received substantial theoretical consideration (see [13, 14, 19] for surveys and recent results) as well as various experimental DNA implementations [6, 10, 20].

In this paper we focus on a specific generalization of the standard 2-handed tile self-assembly model (2HAM) in which we permit initial assemblies to consist of *prebuilt* assemblies of more than one tile. The motivation for studying such a generalization is strong. First, some of the most successful implementations of algorithmic DNA self-assembly utilize a combination of singleton DNA tiles mixed with larger prebuilt assemblies. For example, the experimental implementations of DNA tile counters [10] and the 21 DNA tile circuits

implemented in [20] both utilize a combination of single tiles seeded with a larger prebuilt DNA origami structure encoding the program input. In [10], they additionally include a mix of singleton and prebuilt domino assemblies among the system's tile set. Second, the inclusion of prebuilt shapes of different geometries and sizes allows for the potential application of *steric hindrance*, in which geometric blocking of potential attachments is used to control algorithmic growth, as seen in the theoretical works of [5, 11, 12], and the experimental works of [9, 18]. These examples suggest that consideration of shapes more general than uniform single squares has the promise to allow for improved computational power and efficiency of self-assembled systems.

Given the importance of understanding self-assembly with prebuilt initial assemblies, we consider the complexity of two fundamental computational questions related to verifying the correctness of such systems. First is the *Producibility* problem, which asks if a given tile system can build/produce a given assembly. The second is the *Unique Assembly Verification* (UAV) problem, which asks if a given tile system *uniquely* produces a given assembly, i.e., produces the assembly and nothing else provided sufficient assembly time. For the producibility problem, the 2HAM with just single-tile initial assemblies has a polynomial time solution [7], whereas we show NP-completeness when prebuilt assemblies are permitted. In the case of the UAV problem with singleton tile initial assemblies, the problem resides in coNP [3] for a constant-bounded temperature threshold and is coNP-complete for larger temperature thresholds [15], whereas we show coNP^{NP}-completeness with prebuilt assemblies. In both scenarios, our hardness results hold even for prebuilt assemblies of a bounded $\mathcal{O}(1)$ size and $\mathcal{O}(1)$-bounded temperature thresholds. We accompany these results with a preliminary exploration of the producibility and UAV problems when restricted to 1-dimensional *linear* assemblies with pre-built assemblies, and provide polynomial time solutions.

1.1 Previous work

The model used here differs from the polyTAM model of [11] in that the set of starting elements in our system are defined as assemblies made up of multiple tiles. In the polyTAM, the set of elements are single tiles that are allowed to be larger than a unit square. We are attempting to model the situations where smaller components may have preassembled into larger structures prior to being introduced to the system. This is more similar to the staged model of self-assembly however in that model all the bins are usually assumed to have the same temperature so any assemblies built in early stages must be producible. Here we only require that the input assemblies are stable.

Verification problems have been well-studied in many models of Tile Self-Assembly. In the Abstract Tile Assembly model (aTAM), both the producibility and UAV problem are solvable in polynomial time [1]. When allowing negative or repulsive glues the UAV problem becomes undecidable due to detachment [8], but when restricted to growth-only systems (no detachment can occur) the problem is coNP-complete [4]. Producibility verification in the 2-handed assembly model at any temperature, along with UAV for temperature 1, are both solvable in polynomial time [7]. Membership in the class coNP for general 2HAM systems was shown in [3] along with a hardness result showing coNP-completeness when one step into the 3rd dimension is allowed. By allowing the temperature to be a part of the input UAV has been shown to be coNP-complete [15] even in 2 dimensions. More powerful generalizations have shown an increase in complexity of the UAV problem such as in Tile Automata which merges ideas from Cellular Automata and the 2HAM. This problem was shown to be coNP^{NP}-complete even with the restrictions of Freezing (A tile only changes states a finite number of times) and growth only (no detachment).

Table 1 Complexity of verifying producibility of an assembly in various models. The Assembly Size column indicates the size of the assemblies in the initial assembly set. Previous work has studied the case when only single tiles are allowed. Our results allow for up to constant sized assemblies.

Model	Input Assembly Size	Temperature	Result	Reference
aTAM	Single Tiles	Variable	P	[1]
2HAM	Single Tiles	Variable	P	[7]
2HAM	Constant	2	NP-complete	Thm. 3

Table 2 Complexity results of Unique Assembly Verification in the aTAM and the 2HAM. UAV is undecidable in the negative aTAM, but coNP-complete with negative glues if the system never allows detachment (*growth only). **Tile Automata with freezing and growth only restrictions.

Model	Input Assembly	Temperature	Result	Reference
aTAM	Single Tiles	Variable	P	[1]
Neg. aTAM G.O.*	Single Tiles	2	coNP-complete	[4]
2HAM	Single Tiles	1	P	[7]
2HAM	Single Tiles	Constant	coNP	[3]
2HAM	Single Tiles	Variable	coNP-complete	[15]
2HAM 3D	Single Tiles	2	coNP-complete	[3]
Tile Automata**	Single Tiles	2	coNP^{NP}-complete	[2]
2HAM	Constant	2	coNP^{NP}-complete	Thm. 6

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-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. A Two-handed assembly system is an ordered tuple (S, τ) where S is a set of *initial* assemblies and τ is a positive integer parameter called the *temperature*. Each assembly in S must be τ-stable. For a system (S, τ), the set of *producible* 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 $C \in P'_{(S,\tau)}$.

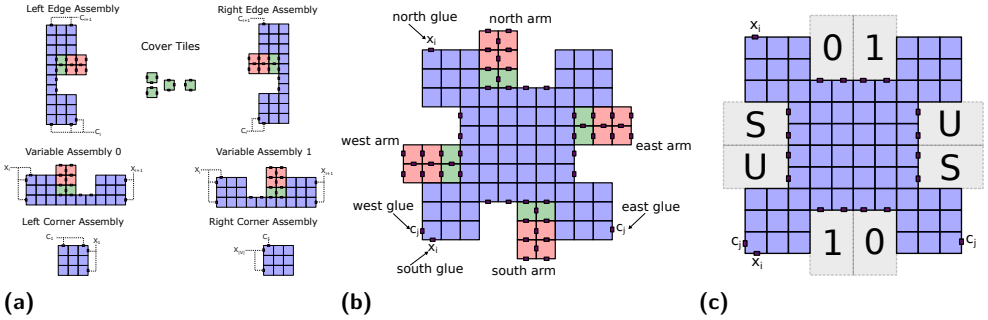

Figure 1 (a) Edge Assemblies used to construct the frame of the assembly. For each clause we include a left and right edge assembly. For each variable we include two variable assemblies representing 0 and 1. We include a single left corner assembly along with a right corner assembly. The filler tiles are used to fill in holes between attached macroblocks. (b) A single macro block $m_{i,j}(0, U, U)$ with outer glues labeled. The north and south glues connect to other macro blocks that represent the same variable. The east and west glues connects to other macroblocks that represent the same clause. (c) Arm position labels on a macroblock. Opposite sides have complementary values to allow for attachment.

A producible assembly is *terminal* provided it is not τ-combinable with any other producible assembly, and the set of all terminal assemblies of a system (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 assemblies that cannot grow any further. An assembly A is *uniquely produced* if $P_{(S,\tau)} = \{A\}$ and for each $B \in P'_{(S,\tau)}$ $B \sqsubseteq A$.

▶ **Definition 1** (Producibility Problem). *Given a 2HAM system $\Gamma = (S, \tau)$ and an assembly A, is A a producible assembly of Γ?*

▶ **Definition 2** (Unique Assembly Verification Problem (UAV)). *Given a 2HAM system $\Gamma = (S, \tau)$ and an assembly A, is A uniquely produced by Γ?*

3 Producibility Hardness

In this section we show that the producibility problem is NP-complete if the initial set of assemblies may include assemblies larger than singleton tiles. The hardness is derived by reducing from 3SAT and holds even if assembly size and system temperature is bounded by a constant. Our construction extends to the seeded *abstract tile assembly model* where there is a seed tile, and elements of the tile set (in this case assembly set) attach one at a time to the growing seed. See appendix for details.

3.1 Overview

We reduce from 3SAT, which asks whether a given 3CNF formula ϕ is satisfiable. Let $|V|$ and $|C|$ be the number of variables and clauses in ϕ, respectively. The reduction creates an instance of producibility (Γ, A) with $\tau = 2$, such that Γ produces the target assembly A iff ϕ is satisfiable. The target assembly is a rectangle shown and described in Figure 4a. The assemblies in the reduction can be divided into two groups: edge assemblies (Figure 1a) and macroblocks (Figure 1b). There are two variable assemblies for each bit that each correspond to an assignment of 0 or 1 based on the position of the 3×2 *arm* section of the assembly

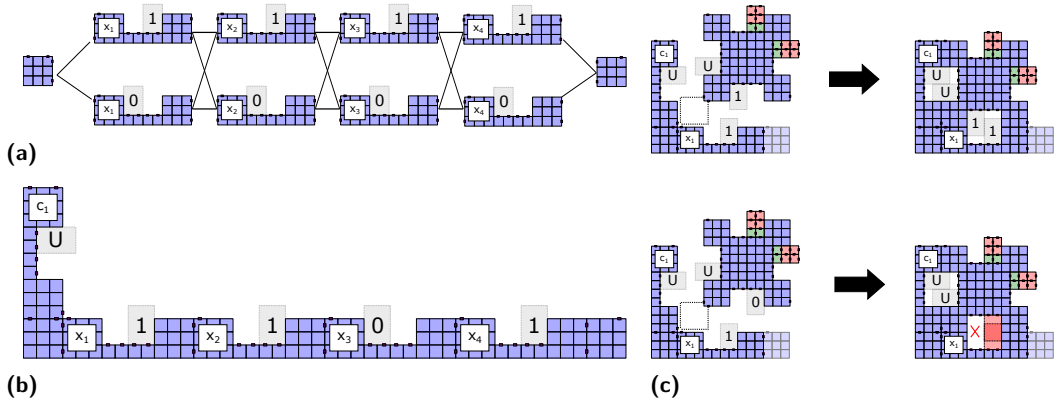

Figure 2 (a) Variable gadgets can non-deterministically build a frame assembly for each possible assignment to ϕ. (b) Example of the assembly made for assignment 1101 with a single left edge assembly attached. (c) If the macroblock has complimentary arm positions it will be able to attach to the frame assembly. If the macroblocks do not have matching arm positions the attachment will be geometrically blocked and cannot occur.

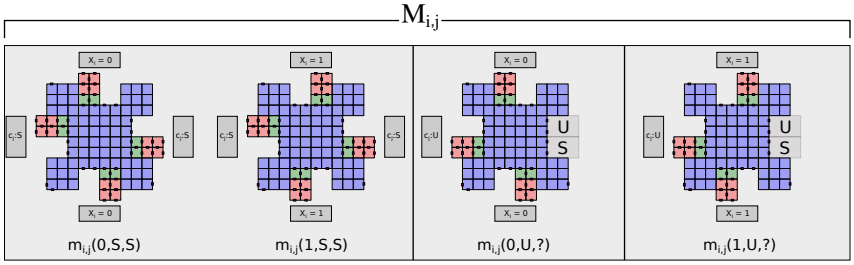

Figure 3 Possible Macroblocks that make up $\mathcal{M}_{i,j}$. Arm positions represent the value assigned to x_i and whether or not c_j has been satisfied. There will always be 4 macroblocks in each set. The left pair of macroblocks are always included and will attach if a clause is already satisfied. The remaining macroblocks attach if the clause is not yet satisfied, and their arm positions depend on ϕ. If the positive literal x_i is in c_j, $m_{i,j}(1, U, S) \in \mathcal{M}_{i,j}$, otherwise $m_{i,j}(1, U, U) \in \mathcal{M}_{i,j}$. If the negative literal $\bar{x_i}$ is in c_j, $m_{i,j}(0, U, S) \in \mathcal{M}_{i,j}$, otherwise $m_{i,j}(0, U, U) \in \mathcal{M}_{i,j}$.

(green and red tiles in the figures). As shown in Figure 1c, each arm position represents a certain value. For vertical arms, the position represents either 0 or 1. For horizontal arms, their position is either U or S, meaning unsatisfied or satisfied.

These assemblies will combine to form an assembly for each possible assignment to ϕ (Figure 2a). We include a unique left edge assembly for each clause whose arm is always in the top position representing that the clause is currently not yet satisfied. A left edge assembly can attach to an assembly that encodes an assignment to ϕ. This starts to form an L shaped frame assembly as shown in Figure 2b. Macroblocks may attach to this frame if it has complementary arms to the currently growing frame assembly. As shown in Figure 2c, a macroblock can attach using two strength-1 glues on its east and south sides. If the arms have complimentary positions the attachment will be able to take place. However, if the arms overlaps, the macroblock is geometrically blocked from attaching, and thus not allowed.

The final challenge for designing this reduction is there must exist a single assembly that is produced regardless of the satisfying assignment. This means we must hide the values passed between macroblocks. We do this by including a certain subset of the tiles which will fill any spaces left between macroblocks.

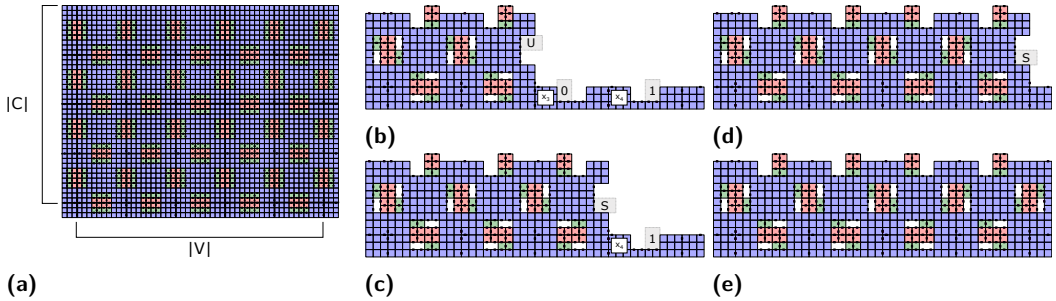

Figure 4 (a) Target assembly for producibility in the 2HAM with prebuilt assemblies. Target assembly is a $10|C|+3$ by $10|V|+6$ rectangle. Smaller rectangles between tiles denote strength-1 glues. Glues between blue tiles are not shown. Each blue tile shares a strength-2 glue with neighboring blue tiles. The exceptions are tiles separated by the thicker borders that do not share a glue unless shown. (b) A frame assembly with macroblocks attached. Here, $c_1 = \overline{x_1} \vee \overline{x_3} \vee x_4$. (c) Here $x_3 = 0$ satisfies the clause so this macroblock that attaches has its arm in the S position. (d) The macroblocks that attach after always have their arms in the S position. (e) The right edge assembly can attach to the last macroblock since its arm position is in on the S position completing the row.

3.2 Macroblocks

A single macroblock can be seen in Figure 1b and has two parts: the body that contains glues to allow attachment (blue), and four arms which encode ϕ (green/red). Each arm on the macroblock encodes a single bit of information by being in one of two positions. We call these positions "0" and "1" for the north/south arms and "U" and "S" (unsatisfied/satisfied) for the east/west arms (Figure 1c).

We denote a macroblock representing a variable/clause pair (v_i, c_j) by its glues and arm positions as $m_{i,j}(b, w, e)$ where $b \in \{0, 1\}$ is the position of the north/south arm, $w \in \{U, S\}$ is the position of the west arm, and $e \in \{U, S\}$ is the position of the east arm. Each macroblock has a single strength-1 glue on each side (macroblocks representing the last clause do not contain glues or arms on their north side). The glues indicate which variable and clause pair this macroblock represents. The north and south glues relate to the variable, and the east and west glues relate to the clause. The south and west glues allow for cooperative attachment to an assembly that already contains macroblocks $m_{i-1,j}$ and $m_{i,j-1}$. The north and east glues allow for attachment of the next macroblocks.

Each variable/clause pair (v_i, c_j) has a set $\mathcal{M}_{i,j}$ of four macroblocks associated with it (shown in Figure 3). The exact macroblocks that are included depends on whether x_i or $\bar{x_i}$ is present in the j^{th} clause. The macroblocks $m_{i,j}(0, S, S)$ and $m_{i,j}(1, S, S)$ are always included since the assignment of a variable can not change a clause from being satisfied to unsatisfied. If the the positive literal v_i appears in c_j, then we include the macroblocks $m_{i,j}(1, U, S)$, or $m_{i,j}(1, U, U)$ if it does not. If the negative literal $\bar{v_i}$ appears is in c_j we include the macroblock $m_{i,j}(0, U, S)$, or $m_{i,j}(0, U, U)$ if it does not.

3.3 Computing Clauses

The left edge assembly starts with an arm in the U position. Macroblocks maintain this position (Figure 4b) until a macroblock attaches that satisfies the clause, and changes the arm to an S position (Figure 4c). Once a row of the assembly is complete (Figure 4d), if the horizontal arm is in the satisfied position, the right edge assembly can attach cooperatively and complete the row (Figure 4e). For a right edge assembly to attach, the clause must be

satisfied and the assembly below it (either another right edge assembly or the right corner assembly) must attach as well. Thus, the right edge assembly only attaches if the clause it represents is satisfied.

Each row has multiple size-2 holes between macroblocks. Filler tiles cannot attach to macroblocks on their own due to the temperature of the system. However, once two macroblocks are attached, the tiles can cooperatively bind across the hole with strength-1 glues from each side (Figure 5b). As shown in Figure 5c, this hides the previously used arm positions. The exposed northern arms also continue to encode the input string so the next clause can be computed after the attachment of the next left edge assembly.

▶ **Theorem 3.** *The producibility problem in the 2HAM with prebuilt assemblies of size $\mathcal{O}(1)$ is NP-complete with $\tau = 2$ and an initial assembly set containing $\mathcal{O}(|V||C|)$ distinct assemblies.*

Proof. For membership in the class NP consider an assembly tree υ for a producible assembly A. To verify A is producible we may use υ as a certificate. If υ is a valid tree (each combination is legal) and each leaf is in the input set I, then A is producible.

To show NP-hardness we reduce from 3SAT. Given a formula ϕ in 3CNF form with $|V|$ variables and $|C|$ clauses. Our target assembly A is the rectangle described above made from macroblocks and edge assemblies with each of the spaces between arms completely filled with tiles. The input assembly set includes I, $|C|$ left edge assemblies with their arm in the unsatisfied position, and $|C|$ right edge assemblies in their satisfied position. For each variable, two input assemblies representing 0 and 1 assignments are included. The clauses are encoded by the selection of macroblock arm combinations. A set of 4 macroblocks are included in I for each variable and clause combination, so there are a total of $\mathcal{O}(|V||C|)$ macroblocks. This results in a total initial assembly set size of $\mathcal{O}(|V||C|)$, and each assembly is constant sized.

The starting assemblies, I, will grow into A if and only if ϕ is satisfiable. If ϕ is satisfiable by some assignment x, then the 'L' shaped frame assembly representing x will grow by attaching macroblocks. Since x satisfies ϕ each clause will eventually have its arm position changed to satisfied allowing for all the right edge assemblies to attach. The single tiles will then fill in the spaces to complete A. If A is producible then there exists some 'L' shaped frame assembled that grew into A. The only way this frame could have grown into A is if the position of the arms on the input assemblies represented a string x that satisfied each clause meaning ϕ is satisfiable. ◀

4 Unique Assembly Verification is coNP$^{\text{NP}}$-complete

In this section we show coNP$^{\text{NP}}$-completeness of the Unique Assembly verification problem in the 2HAM with constant sized prebuilt assemblies. We start by proving UAV is in the class of problems solvable by a nondeterministic algorithm with access to an oracle for a problem in the class NP. We then show hardness by reducing from ∀∃SAT. This is an extension of the reduction shown in the previous section, and further, we utilize similar techniques used in previous reductions in the 2HAM and Tile Automata [2, 16].

▶ **Lemma 4.** *The Unique Assembly Verification problem in the 2HAM with prebuilt assemblies is in coNP^{NP}*

Proof. Given an instance (Γ, A), refer to a "rogue assembly" R as a producible assembly in the system Γ that is either (1) not a subassembly of the target assembly A, $R \not\sqsubseteq A$, or (2) a strict subassembly of A and terminal, i.e., $R \sqsubset A$ and $R \in P_{(S,\tau)}$. The following nondeterministic algorithm solves UAV.

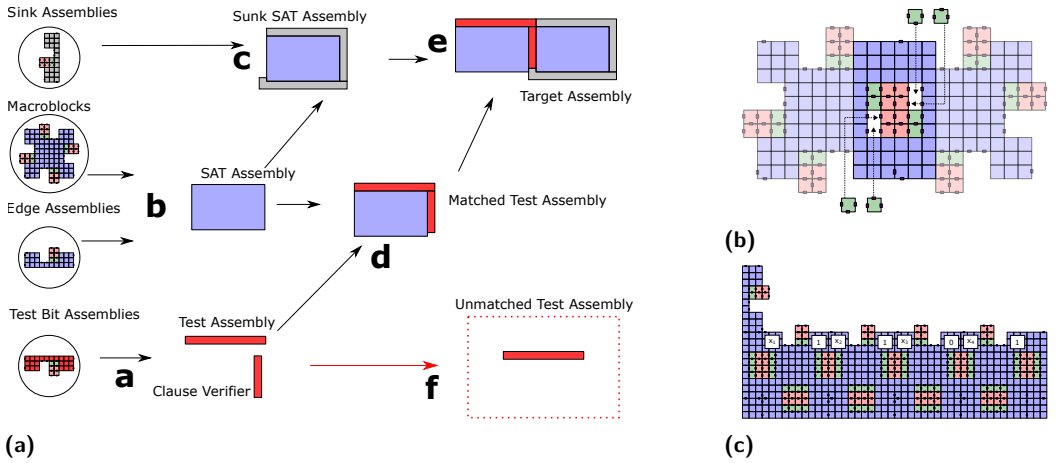

Figure 5 (a.a) Test bit assemblies come together to build a test assembly for all possible assignments of the variables in X. Clause Verifier assemblies may attach to SAT assemblies that satisfied all clauses. (a.b) Macroblocks and edge assemblies from the previous reduction are used to create SAT assemblies that evaluate the formula for every assignment of all the variables. (a.c) The sink assemblies begin attaching to SAT assemblies, ensuring they all grow into the target assembly. (a.d) A test assembly will attach to a SAT assembly that satisfies the formula and has matching assignments to the variables in X. (a.e) Matched test assemblies and sunk SAT assemblies attach to each other forming the target assembly. (a.f) Any test assembly that does not find a SAT assembly to attach to is unmatched and terminal. If any unmatched test assemblies exist, the instance of UAV is false. (b) Once two macroblocks attach, the green filler tiles are able to cooperatively attach using one glue on the macroblock, and the other glue from the red tiles of the arms from the other macroblock. The filling process hides the information that was passed. (c) Another left edge assembly may attach above the first. Since the north arms of macroblocks encode the variable assignment, the second clause may be computed in the same way as the first.

1. Nondeterministically build an assembly B of size $|B| \leq 2|A|$.
2. If B is a rogue assembly, reject.

It suffices to check all assemblies B up to size $2|A|$ since any assembly of size $> 2|A|$ must have been built from at least one other assembly B', s.t. $|A| < |B'| \leq 2|A|$. B' is a rogue assembly itself and will be accounted for in a different branch of the computation. It remains to be checked whether B is a rogue assembly. The first condition can be verified in polynomial time by checking if B is a subassembly of A. The second condition can be checked using an NP oracle that answers the following: "Does there exists an assembly C, $|C| \leq |A|$, such that C can attach to B?". This problem can be solved by an NP machine that nondeterministically builds an assembly C up to size A and attempts to attach it to B. If any C can attach to B, B is not terminal. If any branch finds a rogue assembly, the co-nondeterministic machine will reject. ◀

4.1 Reduction Overview

▶ **Definition 5** (∀∃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 $\phi(X, Y) = 1$?*

We show this problem is coNP^{NP}-hard by reducing from ∀∃SAT. An overview of the important assemblies and processes are shown in Figure 5a. The same construction used in the previous reduction is used to create exponentially many "SAT assemblies", each of

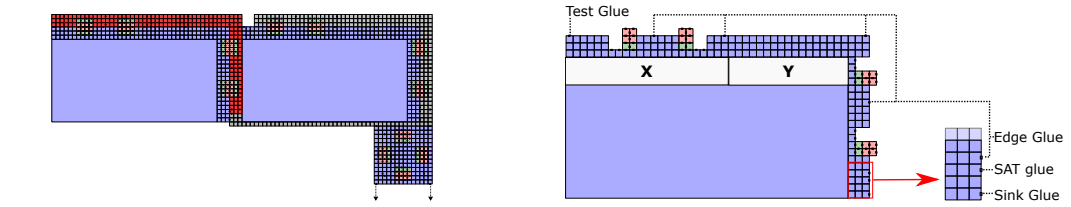

Figure 6 (a) Target Assembly for UAV. Assembly can be divided into three parts, Test assembly with satisfied SAT assembly, Sunk SAT assembly, and a column of clean up frames (b) SAT assembly. Built using the previous reduction with additional northern arms for the variables in X. Also we do not add right edge assemblies an arm is exposed which shows whether a clause has been satisfied.

which evaluates the Boolean formula on one of its input assignments. A SAT assembly is shown in Figure 6b. We do not include right edge assemblies so SAT assemblies that have finished computing will have exposed arms on their right side denoting whether or not each clause was satisfied. The assembly has exposed arms to the north above variables in X that represent their assigned values. Variables in Y will still have no arms on their north side. We construct a test assembly (Figure 7a) for each possible assignment to the variables in X (Figure 8a) along with a *clause verifier* assembly. The clause verifier assemblies are made of right edge assemblies with their arms in the S position. Each test assembly can attach to any SAT assembly with matching assignments to X (complimentary arm positions) if the clause verifier has already attached. Other assemblies are included that "sink" all assemblies to the target except for test assemblies that did not attach to a SAT assembly. Test assemblies may build into the target assembly if and only if they find a matching SAT assembly. The key question about the system is "For all test assemblies, does there exist a compatible SAT assembly where all clauses evaluated to true?"

This system uniquely constructs the target assembly if and only if the instance of $\forall\exists$SAT is true. If the instance is false, there exists an assignment to the variables in X where no satisfying assignment of Y exists. In this case, the test assembly representing that assignment will be terminal, which means the system does not uniquely produce the target assembly.

4.2 SAT Assembly

We use the assemblies from the previous reduction to compute all satisfying assignments to ϕ. We use the same macroblocks, input assemblies, and left edge assemblies, however, we do not include any right edge assemblies or the right corner assembly. A frame assembly is constructed for each assignment to ϕ. The assembly process then computes whether or not the assignment satisfies the clauses in the same way by attaching macroblocks with matching geometry. In this construction we mark the assignment to variables in X by including northern arms to the top most macroblocks for those variables instead of omitting them as in the previous construction. This results in a final assembly that has its assignment to X and the computed value of each clause being encoded in the exposed arm positions.

The assembly can be seen in Figure 6b with glues labeled. The exposed glues all have strength-2. In the bottom right corner of the assembly, the last variable assembly has two glues. The bottom glue is called the sink glue and this is one of the glues the sink assembly uses to attach to a SAT assembly. The glue above it, called the SAT glue, is used by both the left sink base assembly and the test assembly. The next glue appears on each macroblock with an exposed arm, and the northern side of the rightmost macroblock. This glue allows

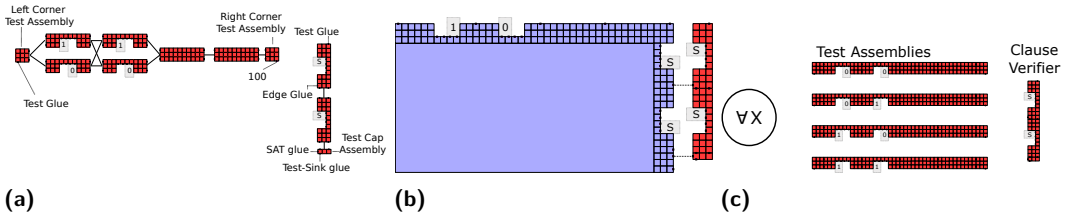

Figure 7 (a) Test bit assemblies nondeterministically construct a test assembly for each assignment to the variables in X. Right edge assemblies with arms in the satisfied positions are used to build the clause verifier. Only the north most edge assembly has an edge glue to allow a clause verifier to attach to a SAT assembly. A test assembly may attach to a SAT assembly with a clause checker attached using the test glues on their south side. The Test-Sink glue is used to cooperatively attach to a sink assembly once the test assembly is matched with a SAT assembly. (b) A clause verifier may attach to SAT assemblies that have their arms in the satisfied position. (c) Test Assemblies that are created for a X that contains 2 variables along with the single clause verifier.

for sink assemblies to attach and cover exposed arms to reach the target assembly. The test glue is the last glue on this assembly and appears in the top right corner. The test assembly uses this glue with the SAT glue to attach to a SAT assembly with matching geometry.

4.3 Test Assembly

A test assembly is constructed for each possible assignment to X starting from constant sized test bit assemblies shown in Figure 7a. For each variable in X, we include two test bit assemblies. For each each variable in Y, we include a blank test bit assembly, which is a 3×10 rectangle. This row is capped by left and right corner test assemblies. The left and right corner assemblies have a strength-1 glue. The clause verifier assembly is built using right edge assemblies. These assemblies all have their arms in the satisfied position. The north most assembly has a strength-1 glue on its left side to attach to SAT assemblies and another on its north side to allow test assemblies to attach. They connect downward to a 1×3 test cap assembly. The test cap has the strength-1 SAT glue on its left side to allow a test assembly to cooperatively bind to a SAT assembly with all arms in the satisfied position (Figure 7b). On its south side it has the test-sink glue which will be used to attach to a sink assembly cooperatively once a SAT assembly is found. This assembly process creates a test assembly for each possible assignment to the variables in X. The example of test assemblies for an instance where $|X| = 2$ and $|Y| = 2$ can be seen in Figure 7c.

The test assembly has two exposed glues on opposite ends of the assembly. Thus, the assembly cannot attach to a SAT assembly until it is completely constructed and a clause verifier assembly has attached. A clause verifier assembly may only attach to SAT assembly with matching arm positions, i.e., SAT assemblies that satisfied all clauses. A test assembly may only attach to SAT assemblies with a clause verifier and that have the same assignment to X. A test assembly along with the four possible SAT assemblies it could attach to is shown in Figure 8a. A test assembly is terminal if there does not exist a SAT assembly with the same assignment to X that satisfies all clauses. A terminal test assembly can be seen in 8b. We call these terminal test assemblies *unmatched* test assemblies. Since we construct a SAT assembly for each possible assignment to the formula ϕ, if the test assembly representing a partial assignment x is terminal that means there does not exist a remaining assignment to the variables in Y that satisfies the formula, and causes the instance of UAV to be false.

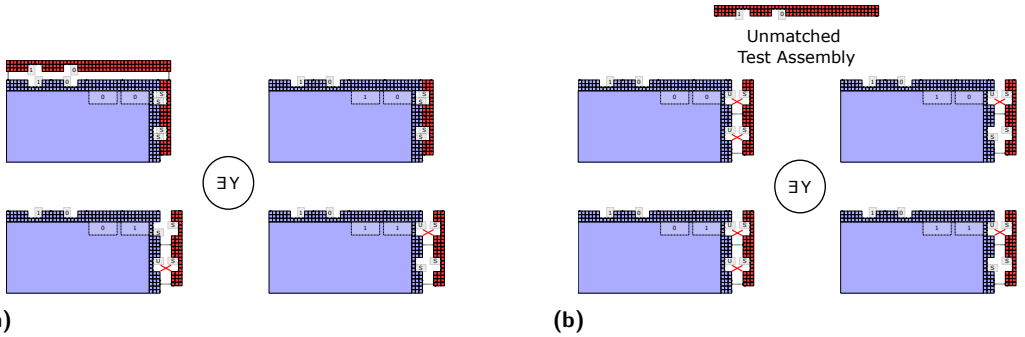

Figure 8 (a) If there exists a satisfying assignment to Y when $X = 10$, the test assembly with those arms can attach to a SAT assembly. (b) If there does not exist a satisfying assignment, all the SAT assemblies will have at least one arm not in the S position that will geometrically block the test assembly's arm. This test assembly will be terminal so the answer to UAV will be no.

4.4 Sink

Since our goal is to design a system that uniquely constructs an assembly when the instance of ∀∃SAT is true, we will *sink* assemblies representing a non-satisfying assignment to the target assembly. This ensures that each assembly (besides unmatched test assemblies) must eventually grow into the target assembly thus sinking all other producible assemblies to our target. We do so via sink assemblies and macroblock frames. The sink assemblies are shown in Figure 9a. The first sink assembly, the right sink base assembly, is similar to the right corner assembly of the previous section but is of height 4. Sink base tiles are single tiles that may attach to the right base assembly on its left side bottom row, which eventually connects to the left base assembly.

The right sink base assembly attaches to any SAT assembly that has not attached to a test assembly using the sink and test-sink glues. We call the attachments that occur after this the *Sink Process*. During the Sink Process, sink edge assemblies attach cooperatively with the right sink base assembly and with the SAT assembly. This allows for the sink edge assemblies to attach one-by-one and "cover" each exposed arm on the SAT assembly. The last sink edge (northern most) is slightly longer to allow for the horizontal sink edges to attach across the top of the assembly. The left sink base assembly cooperatively attaches to test assemblies that have already attached to SAT assemblies using the SAT glue on its right side and the test-sink glue on its north side.

The last assembly type that has not been accounted for in the final assembly are any unused macroblocks. In the previous reduction, any unused macroblocks are terminal since they are never used in the computation. In this reduction, we cannot have any other terminal assembly, so these must be included in our target assembly. We do this by adding frames to store the macroblocks. For each macroblock we include in our input set, we also include a clean up frame (Figure 9b). Any macroblock is now not terminal as it can attach to the clean up frame. These frames are enumerated and attach in order to the south side of the sink assembly in a single column (Figure 9c).

▶ **Theorem 6.** *The Unique Assembly Verification problem in the 2HAM with prebuilt assemblies of size $\mathcal{O}(1)$ is coNP^{NP}-complete with an initial assembly set size of $\mathcal{O}(|V||C|)$ and $\tau = 2$.*

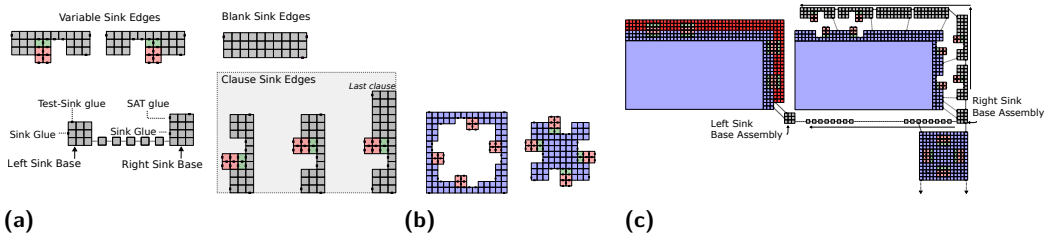

Figure 9 (a) The set of sink assemblies. The sink glue and test-sink glue are utilized for cooperative attachment. (b) For each macroblock we also include a frame so none of the macroblocks are terminal. These frames attach to each other in order at the bottom of the sink assembly. (c) The process of sinking all assemblies towards the target assembly.

Proof. We show membership in Lemma 4. Given an instance of ∀∃SAT with a formula $\phi(x_1, x_2, \ldots, x_n)$ we create a 2HAM system S that uniquely assembles a target assembly S if and only if the instance of ∀∃SAT is true. If the instance of ∀∃SAT is false then S will produce a test assembly that is terminal.

The construction of SAT assemblies begins with combinations of variable and edge assemblies to produce an L shaped frame assembly for each possible assignment to ϕ. The output of ϕ on each assignment is then computed by the attachment of macroblocks that encode the clauses of ϕ, and producing a SAT assembly. The SAT assemblies expose arms representing the assignment of the variables in X, as well as arms that represent whether each clause has been satisfied. From the included prebuilt assemblies, a test assembly is produced for each possible assignment to X. Due to their arm positions, they may only attach to SAT assemblies that have a matching assignment to X and that represent an assignment that satisfies every clause.

Each assembly besides unmatched test assemblies will sink to the target assembly. Every prebuilt assembly that was designed for building a SAT assembly will be used in the construction of at least one SAT assembly, besides the macroblocks. In some cases it is possible for a macroblock to not be able to attach to any SAT assembly. To account for this this we include a macroblock frame for each macroblock to attach to, ensuring that no macroblock is terminal. The right sink base assembly may attach to any SAT assembly regardless of arm position so none of the SAT assemblies are terminal. Each sink assembly is used in the process of reaching the target assembly so none are terminal.

A test assembly is only terminal if there does not exist a SAT assembly with matching X arm positions that has each clause satisfied. This means for that assignment to X there does not exist an assignment to Y that satisfied ϕ, and the instance of ∀∃SAT is false.

The new assemblies in this reduction only increase the number of assemblies by a constant factor. The test and sink assemblies only add $\mathcal{O}(|V| + |C|)$ to the input assembly size, and the added macroblock frames are equal to the number of macroblocks, which is constant. ◀

5 1D Verification

Here, we show that the producibility and the UAV problems in one-dimensional 2HAM (all assemblies are of height-1, and tiles only have glues on their left and right sides) with prebuilt assemblies is solvable in polynomial time. Proofs omitted due to space.

Producibility Verification. At a high level, the algorithm constructs a graph of initial assemblies that may be combined to form the target assembly A. An example graph is shown in Figure 10. We add a starting node s that connects to possible leftmost assemblies and a

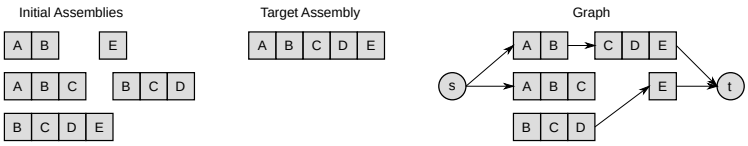

Figure 10 Example instance of Producibility with the graph G created. The path between s and t implies a build sequence to produce the target assembly.

target node t that connects to possible rightmost assemblies. There exists an edge between two assemblies if they may attach as part of an assembly sequence building A. Thus, a path from s to t implies an assembly sequence using those nodes to build the target assembly.

▶ **Theorem 7.** *The producibility problem in the one-dimensional 2HAM with prebuilt assemblies is solvable in polynomial time.*

Unique Assembly Verification. Given an instance of UAV (Γ, A), we first present three conditions that, if checked, provide the answer. If and only if all these conditions are true, the answer to UAV is "yes": 1) A is producible. 2) There does not exist a subassembly $B \sqsubseteq A$ that is terminal. 3) There does not exist a producible assembly $R \not\sqsubseteq A$. Here, R is a rogue assembly since it will never grow into A. As a given system can have an exponential number of assemblies of size $\leq |A|$, we present a lemma showing a limit on the search for a witness that condition 3 is false.

▶ **Lemma 8.** *Let (Γ, A) be an instance of UAV in the one-dimensional 2HAM with prebuilt assemblies such that every initial assembly is a subassembly of A, and A is producible. If there exists a rogue assembly $R \not\sqsubseteq A$, then there exists a rogue assembly R' that can be produced by combining two assemblies R'_1, R'_2 that are subassemblies of A.*

Utilizing this lemma, we search only the subassemblies of our target assembly A to verify the third condition. 1D assemblies only have a polynomial number of subassemblies, so this allows us to check both the second and third condition. Combining these two steps with the polynomial time producibility algorithm from above, we solve UAV in polynomial time.

▶ **Theorem 9.** *The Unique Assembly Verification problem in the one-dimensional 2HAM with prebuilt assemblies is solvable in polynomial time.*

6 Future Work

Here we showed NP-completeness of the producibility problem and coNP^{NP}-completeness of UAV in the 2HAM with constant-sized prebuilt assemblies with $\tau = 2$. The non-cooperative versions ($\tau = 1$) of many models see a drop in complexity for these problems and have been proven to be incapable of universal computation. However, the authors of [11] show that even without cooperative binding the, Polyomino Tile Assembly Model (tiles may be larger polyominoes instead of only unit squares) is capable of universal computation. Since the polyominoes in this model are similar to prebuilt macroblocks, perhaps the same results may be proven in the 2HAM with prebuilt assemblies and $\tau = 1$.

References

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

2 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.

3 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert T. Schweller, Scott M Summers, and Andrew Winslow. Two Hands Are Better Than One (up to constant factors): Self-Assembly In The 2HAM vs. aTAM. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 172–184. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.

4 Angel A Cantu, Austin Luchsinger, Robert Schweller, and Tim Wylie. Covert computation in self-assembled circuits. *Algorithmica*, 83(2):531–552, 2021.

5 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.

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

7 David Doty. Producibility in hierarchical self-assembly. In Oscar H. Ibarra, Lila Kari, and Steffen Kopecki, editors, *Unconventional Computation and Natural Computation*, pages 142–154, Cham, 2014. Springer International Publishing.

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

9 Masayuki Endo, Tsutomu Sugita, Yousuke Katsuda, Kumi Hidaka, and Hiroshi Sugiyama. Programmed-assembly system using DNA jigsaw pieces. *Chemistry: A European Journal*, pages 5362–5368, 2010.

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

11 Sándor P Fekete, Jacob Hendricks, Matthew J Patitz, Trent A Rogers, and Robert T Schweller. Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 148–167. SIAM, 2014.

12 Bin Fu, Matthew J Patitz, Robert T Schweller, and Robert Sheline. Self-assembly with geometric tiles. In *International Colloquium on Automata, Languages, and Programming*, pages 714–725. Springer, 2012.

13 Pierre-Étienne Meunier, Damien Regnault, and Damien Woods. The program-size complexity of self-assembled paths. In *STOC: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 727–737. Association for Computing Machinery, 2020. doi:10.1145/3357713.3384263.

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

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

18 Sungwook Woo and Paul W.K. Rothemund. Stacking bonds: Programming molecular recognition based on the geometry of DNA nanostructures. *Nature Chemistry*, 3:620–627, 2011.

19 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):16–22, 2013.

20 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.