Covert Computation in the Prebuilt aTAM*

Timothy Gomez† Robert Schweller‡ Tim Wylie‡

Abstract

We prove several results related to the concept of hidden computation in the most well-known model of self-assembly, the Abstract Tile-Assembly Model (aTAM). Previous work showed that the aTAM, with negative glues and no detachment, or in 3D, is capable of covert computation. Without negative glues, the aTAM is still capable of covert computation, but it only seems possible with exponential-sized output assemblies. Here, we show that with a constant number of constant-sized prebuilt assemblies, covert computation is possible. Further, we use this to show that the fundamental self-assembly problem of Unique Assembly Verification (UAV) in the prebuilt aTAM is coNP-complete, which is in contrast with UAV in the standard aTAM, which is polynomial. Finally, we look at the polyTAM and prove that Producibility is in P and UAV is FPT with respect to the size of the polytiles in the tile set.

1 Introduction

With the ability to manufacture nanoscale structures and to use DNA as building blocks for structures [24] or for data storage [11], there has been a great increase in the need to process and compute information at the same level. Thus, the study of self-assembling computation has been an important and active area of research over the last two decades.

Designing self-assembling systems that compute functions is an active and well-studied area of computational geometry and biology [5, 19]. The ability to craft monomers capable of placing themselves, especially when doing precision construction and computation at scales where conventional tools are incapable of operating, (e.g., the nanoscale) has tremendous power. One of the few downsides to self-assembly computation is that the entire history of the computation is visible. In certain cases, this may be undesirable for privacy or security reasons, which we motivate below. Thus, we build on recent work [3, 7, 8, 10] to explore *covert computation*, where we build Tile Assembly Computers (TACs) designed to obtain the output of computation while obscuring the inputs and computational history. We do this by proving that covert computation is possible in one of the simplest standard models of self-assembly (the Abstract Tile-Assembly Model [26]) if only a constant number of prebuilt assemblies are allowed in the tile set.

1.1 Previous Work

The Abstract Tile-Assembly Model (aTAM) was first introduced in [26] and inherited the ability to perform Turing computation from Wang tiles [25]. Since then, investigation into the model has led in many directions, such as Intrinsic Universality [17, 20], efficient assembly of shapes [22], and parallel computation [6, 21]. Many generalizations have also appeared, such as allowing for RNA tiles that can be deleted [1, 14], multiple stages of growth [7, 13, 15], and even negative glues [10, 16]. The aTAM is powerful because not only can the tile set store information, but work has also gone into using the seed [4], or even the temperature [12, 23], for making systems more complex.

Tile Assembly Computers were defined in [6, 21], and Covert Computation, as defined in the field of self-assembly, was first introduced in 2019 [10] for negative growth-only aTAM. In *negative* variations of tile self-assembly models, tiles are capable of not only attachment to, but also detachment from, an assembly if the assembly has a cut through the bond graph less than the temperature, which might have been introduced by a negative glue. In negative *growth-only* aTAM, the system must be designed so that no tile detachment can occur, even when there may be glues providing a repellent force. The covert computation framework was created to analyze the Unique Assembly Verification (UAV) problem within that model, and showing it to be coNP-complete. Covert computation has been explored in two other models of self-assembly as well: Staged Self-Assembly [7] and Tile Automata [8].

Unique Assembly Verification and Producibility are fundamental problems in the field of self-assembly. In the standard aTAM (no negative glues), the Producibility and UAV problems are solvable in polynomial time [2]. These algorithms generalize to the 3D aTAM as well. However, covert computation *is* possible in the 3D aTAM [3] even though UAV is polynomial. With prebuilt assemblies in the aTAM, Producibility is NP-complete, and UAV is known to be coNP-complete with

*This research was supported in part by National Science Foundation Grant CCF-2329918.

†Massachusetts Institute of Technology, tagomez7@mit.edu

‡University of Texas Rio Grande Valley, {robert.schweller,timothy.wylie}@utrgv.edu

Model	Max Size	Amt. >1	τ	Result	Ref.
aTAM	1	0	τ	P	[2]
Neg$_{GO}$ aTAM*	1	0	2	coNP-c	[10]
3D aTAM	1	0	2	P	[2]
aTAM	38	$O(\|A\|)^{\dagger}$	2	coNP-c	[9]
aTAM	12	8	2	coNP-c	Thm. 1
polyTAM	$O(1)$	any	τ	FPT	Thm. 3

Table 1: Complexity results of Unique Assembly Verification in the aTAM with our result added. All other UAV results have focused on the tile set being singletons. UAV is undecidable in the negative aTAM, but coNP-complete with negative glues if the system never allows detachment (*growth only). τ is the temperature of the system. Max Size refers to the size of the attachable tiles in the tile set, and the Amt. > 1 column is the number of different polyominos greater than a singleton in the system. † This refers to the size of the assembly A that is input to UAV.

a linear number of constant-sized (38) assemblies [9]. A summary of some previous UAV results in the aTAM is shown in Table 1.

1.2 Our Contributions

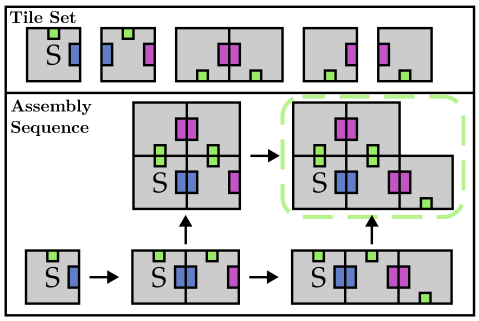
Here, we drastically improve the previous prebuilt aTAM result by proving that, not only is it capable of covert computation, but that UAV, even with only eight prebuilt assemblies of constant size (12), is still coNP-complete. This is different from the polyTAM model [18] since each prebuilt assembly must be built from individual tiles in the system. The covert construction and UAV result are covered in Section 3. Following in Section 4, we show that Producibility in the polyTAM (to show a separation from the prebuilt model) and Unique Assembly Verification are Fixed Parameter Tractable with respect to the size k of the largest polytile.

2 Definitions

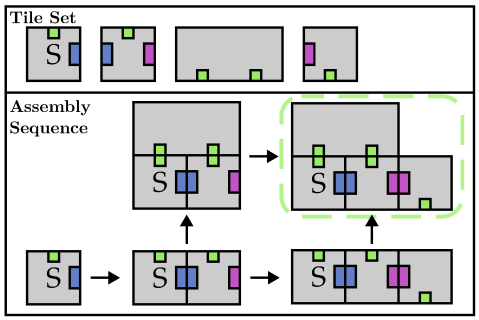
We begin with an overview of the Prebuilt Abstract Tile-Assembly Model, the Polyomino Tile Assembly Model, and then follow with a definition of Tile Assembly Computers and covert computation.

2.1 Abstract Tile Assembly Model

We provide a brief, high-level definition of the Abstract Tile-Assembly Model (aTAM) and refer the reader to [22] for a formal definition. At a high level, the Abstract Tile-Assembly Model (aTAM) uses a set of *tiles* capable of sticking together to construct shapes. These



(a) Prebuilt aTAM Example



(b) polyTAM Example

Figure 1: Example temperature-2 systems with a seed S and a 2×1 tile. The terminal assembly is outlined, matching colors are matching glue labels, and the size of the glue is the strength (1 or 2). (a) Prebuilt aTAM system. (b) polyTAM system.

tiles are typically squares (2D) or cubes (3D) with *glues* on each side where they may attach to one another. A glue is labeled to indicate its type, governing what other tiles it may bond with and the *strength* of the bond. A tile with all of its labels is a *tile type*. A *tile set* contains all the tile types of the system. A single tile may attach at a location if the combined strength of the matching glues is greater than or equal to the *temperature* τ. An *assembly* is a shape made up of one or more combined tiles. Construction is started around a designated *seed* assembly S. Any assembly capable of being made from the seed is called a *producible* assembly. An assembly is *terminal* if no more tiles can attach. A terminal assembly is said to be *uniquely produced* if it is the only terminal assembly that can be made by a tile system. A tile system is formally represented as an ordered triplet $\Gamma = (T, s, \tau)$ of the tile set, seed assembly, and temperature parameter, respectively.

2.2 Prebuilt aTAM and polyTAM

Formal definitions for the prebuilt aTAM and the polyTAM can be found in [9] and [18], respectively. Figure 1 gives a small example aTAM and polyTAM system to illustrate the main concepts and the difference in larger attaching assemblies. Briefly, the prebuilt aTAM generalizes the aTAM by allowing the tile-set to contain pre-assembled, τ-stable macro assemblies that attach

to a growing seed if such attachment yields at least τ strength. Similarly, the polyTAM allows polyomino-shaped tiles in the tile set, but does not require that these polyominos be made up of individual tiles, which fundamentally affects the problems of assembly production and unique assembly production.

2.3 Covert Computation

We provide a brief, informal definition of covert computation, and refer the reader to [10, 21] for a formal definition. Briefly, a tile system computes a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^k$ if there exists a uniform method for encoding an n-bit string x into a seed such that the unique terminal assembly of the system with such seed maps to the k-bit string $f(x)$. A system that computes f is further said to *covertly* compute f if $f(x_1) = f(x_2)$ implies that the final terminal assemblies produced from inputs x_1 and x_2 are guaranteed to be identical. In other words, the assembly produced from the computation of $f(x)$ cannot be used to distinguish which input was used to create it. This includes obfuscating not only the input bit string encoded in the initial seed, but also the entire computational history of the computation. In contrast, standard aTAM Turing machine simulations explicitly encode the initial bit string and each computational step of the computation's history.

3 Covert Computation in the Prebuilt aTAM

Here, we show that covert computation is possible in the 2D aTAM with polynomial assembly size if we allow prebuilt assemblies. We then show that this implies Production is NP-complete and Unique Assembly Verification (UAV) is coNP-complete in the prebuilt aTAM via a reduction from Circuit SAT. Although both of these problem complexities were known for the general problem [9], we greatly lower the size of the prebuilt assemblies from 36 to 12, and we only need 8 different assemblies rather than a number polynomial in the input assembly size. Thus, we have shown the problems can not be parameterized on the prebuilt assembly size (they are paraNP-complete/para-coNP-complete).

The hardness result follows the same basic construction design as used in the construction from [10]. We then follow with the results for a constant number of prebuilt assemblies.

Definition 1 (Planar Circuit SAT) *Instance: A planar directed acyclic graph (DAG) $G = (V, E)$ with n boolean inputs, one output, and all gates are NAND gates (or NOR gates). Every $v \in V$ is either a NAND gate ($deg^-(v) = 2$, $deg^+(v) = 1$) or a fanout ($deg^-(v) = 1$, $deg^+(v) = 2$). The source vertices, $v_i \in V$ s.t. $deg^-(v_i) = 0$ and $1 \leq i \leq n$, are the variables. The sink vertex, $s \in V$ s.t. $deg^+(v_i) = 0$ is the "output" of the boolean circuit.*
Question: Does there exist a setting of the inputs such that the output to the circuit is 1?

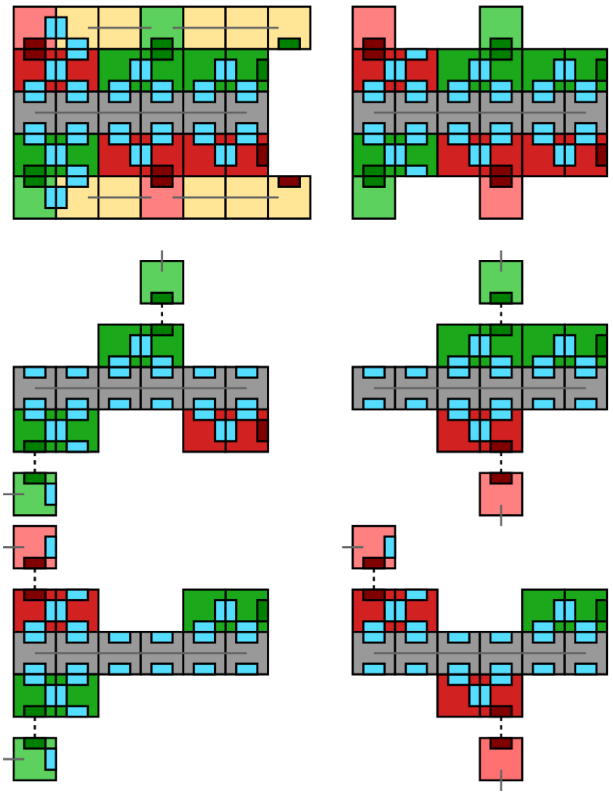


Figure 2: The NAND gate construction. The green, red, and blue glues are all strength one (system strength is 2), and the lines represent unique full-strength glues. The fully constructed gate at the end is shown on the left. The input to any of the gates will be the upper green (True) or lower red (False) tiles on the left side, which each have a strength-1 green/red glue. Depending on which two are present, there are two prebuilt assemblies that can attach: the one with two green tiles on the right, and the one with the reds. These are equivalent to outputting True or False, respectively. In order, there is the full gadget, the 2 True inputs with False output, True/False with True output, False/True with True output, and 2 False inputs with a True output.

3.1 Overview

We use a dual-rail logic implementation of variables and wires to build NAND, FANOUT, and BUFFER gates. As each gadget attaches, it backfills the previously unused wires and gadgets so that the input to a gate is obscured (see Figure 3a).

The main insight for the construction is that a 4×4 block can be used in a prebuilt gadget where the assembly is stable with strength-1 glues, but it is not constructable from a seed without other tiles (see Figure 3b). Thus, only when backfilling are helping tiles there to allow other parts of the gadget to assemble. These

(a) Backfilling (b)

Figure 3: (a) Backfilling with a NAND (gadget A)and NOT (gadget B) connected. Only when the inputs are received will the gadget backfill the other possible input value. (b) The minimum prebuilt assembly that can not be assembled from a seed in a strength-2 system.

4×4 blocks are visible in all gadgets as the cycles using strength-1 blue glues.

Variables and Wires. A single row of tiles represents a true or false signal in the dual-rail logic. These are seen as the input to the gates shown in Figures 2 and 5 where the green tiles are true and the red false.

NAND Gate. The NAND gate is shown in Figure 2. The first image shows the filled in gadget, and the next four show the different prebuilt assemblies that function as the NAND gadget. The red and green tiles on the left side are the wires attaching to the white tiles of the gadget. The two wires are both needed for the gadget to cooperatively attach, but only the prebuilt gadget with the correct output can attach. The lines attaching tiles all have strength-2 glues, and all other blue, green, and red glues have strength-1.

Note that the prebuilt assemblies can not be built with single tile attachments due to the output tiles that are cooperatively attached, but could not have been built in the aTAM. This is also why the other truth value line does not populate when the gadget attaches. The two missing tiles attached with blue glues can only be filled once the other line has backfilled to the gadget.

Finally, we note that each NAND needed in a circuit is a distinct prebuilt assembly (with the four variants) due to needing distinct glues for the inputs. If these were not unique, the circuit would not be built properly.

FANOUT Gate. Figure 5 shows the complete gadget and the two prebuilt assemblies that implement it. The output works similar to the NAND gadget where the other output tiles can only be placed by backfilling.

BUFFER Gate. For the fanout, since we need one rail to backfill before the other rail can go, we create a backfill gadget (Figure 6), that simply forces the backfill to happen behind itself. We can then place these on the output of each fanout rail. These would be needed if you had both outputs of a fanout going into the same NAND as in Figure 7.

3.2 Verification with Prebuilt Assemblies

Theorem 1 *Production and Unique Assembly Verification in the prebuilt aTAM are NP-complete and coNP-complete respectively with only 8 prebuilt assemblies of size-12 or fewer tiles.*

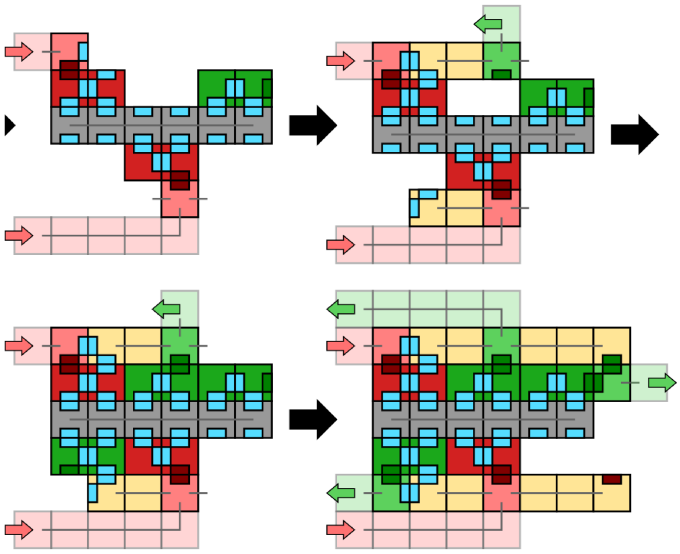


Figure 4: The NAND gadget with two false inputs. The true rails backfill, then the side tiles go around the gadget, but only the prebuilt tiles can continue the signal. The rest of the gadget completes when the false rail is backfilled from the next gadget.

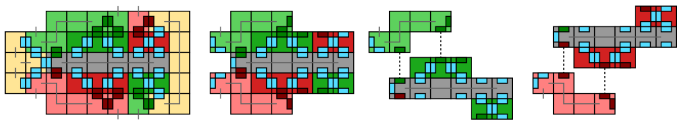


Figure 5: The FANOUT construction. Assumes that the end of the wire has a unique tile indicating a fanout. The fully filled gadget is shown on the left. The center figures shows the prebuilt True FANOUT attaching to a True line, and the end figure shows a prebuilt False FANOUT attaching to a False line.

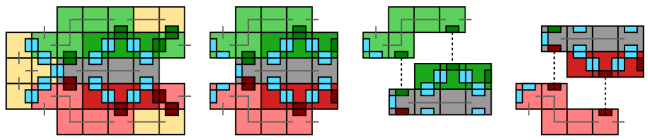


Figure 6: The backstop/buffer gadget only requires one input. This is equivalent to a buffer gate.

Proof. Given any planar Circuit SAT with a DAG made of wires, NANDs, and FANOUTs, we build a prebuilt aTAM system by converting all gates to functional NAND and FANOUT prebuilt assemblies (with unique glues) in the system. The wires are made of unique tiles that can only connect the prebuilt assemblies.

We create the seed assembly as shown in Figure 9 where either true or false prebuilt gadgets can attach to each variable. Each variable has its own version of the true and false assembly that differ by the connecting glue. Thus, all possible seeding truth values can occur.

Once each gadget places, it backfills the previous

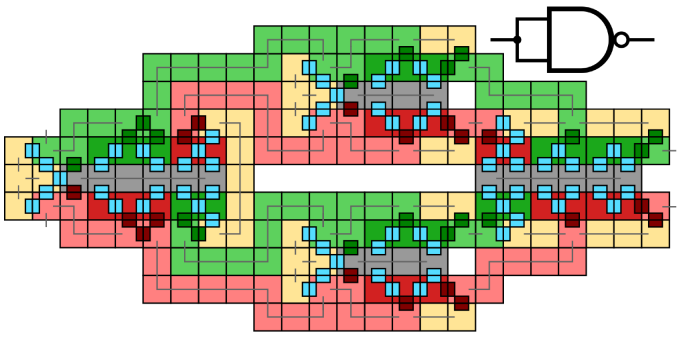


Figure 7: An example usecase where the backstop gadget is needed. Here, a fanout has both outputs going into the two inputs of a NAND. If a false signal was the input to the fanout, without the backstop to force the backfilling to assemble the other output of the fanout, the NAND could never cooperatively attach.

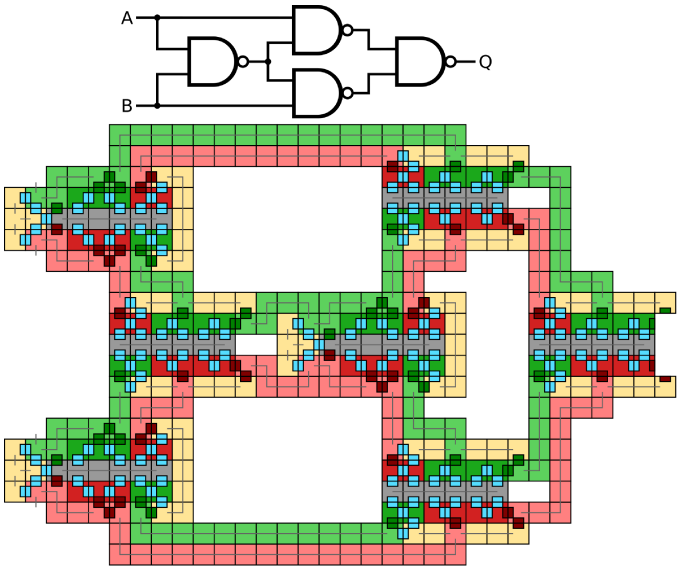


Figure 8: The XOR gate implemented with NANDs and FANOUTs and then shown as a terminal assembly built from the prebuilt assemblies and tiles. Note that in an actual build, only the true or false would be output in the final NAND gadget.

wires and gadgets. Thus, at the end, all terminal assemblies look the same unless some assignment satisfies the circuit sat, which would then have a 'True' output on the final gadget as opposed to a 'False'. □

4 PolyTAM UAV

In this section, we consider the complexity of producibility and UAV in the polyTAM. Unlike the prebuilt aTAM, the polyominos of the polyTAM are not made up of individual tiles. This changes the complexity of producibility and UAV since, in the prebuilt model, multiple distinct layouts of prebuilt assemblies

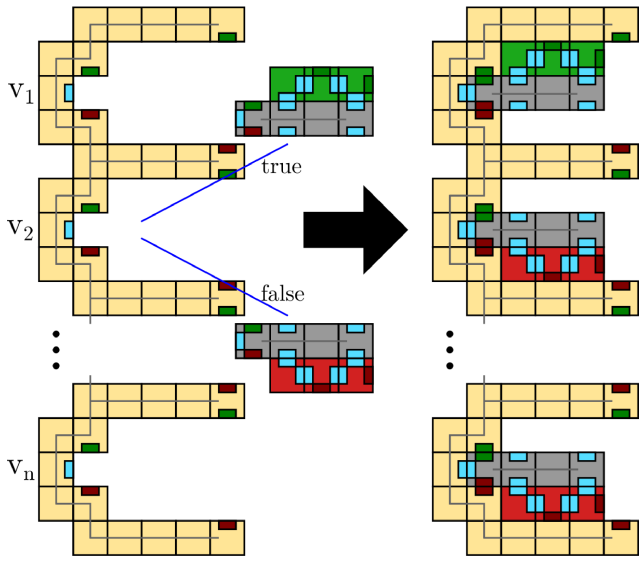


Figure 9: The input seed is a simple linear assembly with variable buffer assemblies attached. The buffer gadgets can be attached nondeterministically with the cooperative input glues. For the Circuit SAT reduction, we must allow all possible assignments. The linear assembly seed with no variables attached can be added, and all possible assignments are feasible.

could come together to form a unique underlying pattern of tiles. However, in the polyTAM, this does not happen, making the problems of producibility and UAV provably easier.

Unique Assembly Verification has not been studied within the literature for the polyTAM, however, a slightly more involved variation of the aTAM UAV algorithm (based on the size of the polyominos in the tile set) can solve the problem [2].

Theorem 2 *Producibility in the PolyTAM is in P.*

Proof. We run the same greedy grow algorithm from [2]. For system $\Gamma = (T, S, \tau)$ and target assembly A, start with the seed assembly $A' = S$. While there exists a poly tile $t \in A$ that may attach to A', attach it. If $A' == A$, then accept, or if we reach a terminal assembly such that $A' \neq A$, then reject.

At a high level, the algorithm correctness still holds because placing a tile never prevents reaching the target and thus, order of placement does not matter. Formally, if $A' \to B$ and $A' \to C$, then $C \to B \bigcup C$. If $A' \to B$ via attaching tile p, then p may still attach to C. All the glues used by p to attach still exists in C because $A' \sqsubseteq C$. Further, the tile q used by $A' \to C$ does not overlap with p since they both exist in A. Thus, we can attach p to C and reach $A' \bigcup \{p\} \bigcup \{q\} = B \bigcup C$. □

The algorithm also generalizes the UAV algorithm for the aTAM from [2]. However, instead of only excluding

a single tile in each loop, we exclude all the neighborhoods of that tile. At a high level, we must do this to make sure we avoid any possible *geometric blocking*.

Theorem 3 *Unique Assembly Verification in the Poly-TAM is solvable in time $2^{O(k)}poly(|T|, |A|)$ where k is the size of the largest polytile. Thus, UAV is FPT with respect to polytile size.*

Proof. For polytile $p \in A$, let $N_A(p, i)$ be the set of polytiles q such that q is within i unit squares of p in A. We refer to a rogue assembly as any assembly R such that either (1) $R \not\subseteq A$ or (2) R is a strict subassembly of A and is terminal. Thus, the existence of a producible rogue assembly is sufficient and necessary for a UAV instance to be false. While in the prebuilt aTAM we had to handle the second case in the polyTAM, we know that if the assembly is producible then no subassembly is terminal via the same argument as in producibility. Placing a tile from A never prevents reaching the target.

The algorithm is as follows: for each polytile $p \in A$ and subsets $Q \subseteq N_A(p, 4k)$, consider the assembly $B = A \setminus p \bigcup Q$. Starting from the seed S, run the producibility algorithm from Theorem 2 and take the maximally produced subassembly $C \sqsubseteq B$. Check if any tile $r \neq p$ can attach to C. If any such tile can attach, then return $R = C \bigcup r$.

It remains to be proven that the returned R is a rogue assembly. If the UAV instance is false, then there exists some producible rogue assembly R'. Since it is producible there exists some $R' \to R$, we can repeat such a process until we reach some non-rogue assembly $B' \to R$. Thus, we can always assume there exists a rogue that is within 1 attachment from some subassembly $B' \sqsubseteq A$. Consider the smallest such B'. Let r be the polytile such that $R = B' \bigcup \{q\}$, which exists since $B' \to R$. Let $\beta_A(r)$ be the subset of polytiles of A that overlap with r, i.e., the tiles that block r. Let $\alpha_{B'}(r)$ be a minimum set of polytiles in B' that r uses to attach to B', i.e., the glue tiles. Now consider some other producible assembly C', as long as $C' \bigcup \alpha_{B'}(r) = \alpha_{B'}(r)$ and $C' \bigcap \beta_A(r) = \emptyset$, i.e., C' allows r to attach. Now we will find some C' in our algorithm and that attaching r to C' forms a rogue assembly. Consider some tiles $e \in A$ such that e and r overlap. Consider the neighborhood $N(e, 4k)$. Note everything in $\alpha_{B'}(r)$ and $\beta_A(r)$ are included in $N(e, 4k)$ as the bounding box containing both r and e is max $2k$ in both dimensions. We need to double the size of the bounding box again since $\beta_A(r)$ and $\beta_A(r)$ are within k distance of r. Thus, since we check all subsets of $N(e, 4k)$, we will find a subset with everything in $\alpha_{B'}(r)$ and nothing in $\beta_A(r)$. Thus, we build some assembly C' that allows r to attach building a rogue assembly. □

5 Conclusion and Future Work

In this paper we showed the prebuilt aTAM is capable of covert computation and used that to improve the hardness result of verification problems to show they only require a constant number of tiles. Thus, in parameterized complexity terminology these problems are paraNP-complete (or para-coNP-complete) with respect to the prebuilt assembly size, which means the problem is NP-hard (coNP-hard) for some exact constant, which is 8 for these problems. This in contrast to the polyTAM where we have an FPT algorithm with respect to the polytile size. These results lead to some interesting directions for future work.

- One open problem is to improve the FPT algorithm. Can we achieve a polynomial time algorithm or is it NP-hard for super-constant sized polytiles?
- In the prebuilt aTAM we have pushed the prebuilt assembly size from 38 to 12. With a modification of the original aTAM UAV algorithm, similar to the FPT algorithm for polyTAM, it seems feasible to show that UAV with prebuilt assemblies of size ≤ 3 is polynomial since size-4 assemblies are the smallest stable assemblies that could exist that is not buildable from singletons (a cycle of 4 tiles with strength-1 glues between them in a $\tau = 2$ system). See Figure 3b. This leaves a complexity gap for prebuilt assemblies of sizes $4 - 11$.
- We only used 8 different prebuilt assemblies. What if only a single prebuilt assembly is allowed?
- Covert computation is already possible in the 3D aTAM, however, we believe that in the 3D prebuilt aTAM, if we modify the covert circuit (from [9]) with a fewer number of smaller sized prebuilt assemblies, covert computation is possible and UAV would be coNP-complete, which is in P without prebuilt assemblies.
- Although not investigated, we may also get some straightforward corollaries from the polyTAM, such as the prebuilt aTAM being capable of universal computation at temperature-1 whenever we have prebuilt assemblies of at least size 3 [18].

References

[1] Z. Abel, N. Benbernou, M. Damian, E. D. Demaine, M. L. Demaine, R. Flatland, S. D. Kominers, and R. Schweller. Shape replication through self-assembly and rnase enzymes. In *Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'10, pages 1045–1064, 2010.

[2] L. M. Adleman, Q. Cheng, A. Goel, M.-D. A. Huang, D. Kempe, P. M. de Espanés, and P. W. K. Rothemund.

Combinatorial optimization problems in self-assembly. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 23–32, 2002.

[3] R. M. Alaniz, D. Caballero, T. Gomez, E. Grizzell, A. Rodriguez, R. Schweller, and T. Wylie. Covert Computation in the Abstract Tile-Assembly Model. In *Symposium on Algorithmic Foundations of Dynamic Networks*, volume 257 of *SAND'23*, pages 12:1–12:17, 2023.

[4] A. Alseth and M. J. Patitz. The need for seed (in the abstract tile assembly model). In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, SODA'23, pages 4540–4589, 2023.

[5] S. Berman, S. P. Fekete, M. J. Patitz, and C. Scheideler. Algorithmic foundations of programmable matter (dagstuhl seminar 18331). In *Dagstuhl Reports*, volume 8. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[6] Y. Brun. Arithmetic computation in the tile assembly model: Addition and multiplication. *Theoretical Comp. Sci.*, 378:17–31, 2007.

[7] D. Caballero, T. Gomez, R. Schweller, and T. Wylie. Covert computation in staged self-assembly: Verification is pspace-complete. In *Proceedings of the 29th European Symposium on Algorithms*, ESA'21, 2021.

[8] D. Caballero, T. Gomez, R. Schweller, and T. Wylie. Verification and computation in restricted tile automata. *Natural Computing*, 2021.

[9] D. Caballero, T. Gomez, R. Schweller, and T. Wylie. Complexity of verification in self-assembly with prebuilt assemblies. *Journal of Computer and System Sciences*, 136:1–16, 2023.

[10] A. A. Cantu, A. Luchsinger, R. T. Schweller, and T. Wylie. Covert computation in self-assembled circuits. *Algorithmica*, 83:531 – 552, 2019.

[11] L. Ceze, J. Nivala, and K. Strauss. Molecular digital data storage using dna. *Nature Reviews Genetics*, 20(8):456–466, 2019.

[12] C. Chalk, A. Luchsinger, R. Schweller, and T. Wylie. Self-assembly of any shape with constant tile types using high temperature. In *Proc. of the 26th Annual European Symposium on Algorithms*, ESA'18, 2018.

[13] C. T. Chalk, E. Martinez, R. T. Schweller, L. Vega, A. Winslow, and T. Wylie. Optimal staged self-assembly of general shapes. *Algorithmica*, 80(4):1383–1409, 2018.

[14] E. Demaine, M. Patitz, R. Schweller, and S. Summers. Self-assembly of arbitrary shapes using rnase enzymes: Meeting the kolmogorov bound with small scale factor. *Symposium on Theoretical Aspects of Computer Science (STACS2011)*, 9, 01 2010.

[15] E. D. Demaine, S. P. Fekete, C. Scheffer, and A. Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Computer Science*, 671:4 – 18, 2017. Computational Self-Assembly.

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

[17] D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, S. M. Summers, and D. Woods. The tile assembly model is intrinsically universal. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 302–310. IEEE, 2012.

[18] S. P. Fekete, J. Hendricks, M. J. Patitz, T. A. Rogers, and R. T. Schweller. Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly. In *Proceedings of the 2015 Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'15, pages 148–167, 2015.

[19] P. W. Frederix, I. Patmanidis, and S. J. Marrink. Molecular simulations of self-assembling bio-inspired supramolecular systems and their connection to experiments. *Chemical Society Reviews*, 47(10):3470–3489, 2018.

[20] D. Hader, A. Koch, M. J. Patitz, and M. Sharp. The impacts of dimensionality, diffusion, and directedness on intrinsic universality in the abstract tile assembly model. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2607–2624. SIAM, 2020.

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

[22] P. W. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 459–468, 2000.

[23] R. Schweller, A. Winslow, and T. Wylie. Complexities for high-temperature two-handed tile self-assembly. In R. Brijder and L. Qian, editors, *DNA Computing and Molecular Programming*, pages 98–109, Cham, 2017. Springer International Publishing.

[24] K. F. Wagenbauer, C. Sigl, and H. Dietz. Gigadalton-scale shape-programmable dna assemblies. *Nature*, 552(7683):78–83, 2017.

[25] H. Wang. Proving theorems by pattern recognition — ii. *The Bell System Technical Journal*, 40(1):1–41, 1961.

[26] E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.