5-2022

Computational Complexity in Tile Self-Assembly

Timothy Gomez
The University of Texas Rio Grande Valley

COMPUTATIONAL COMPLEXITY OF VERIFICATION IN TILE SELF-ASSEMBLY

A Thesis

by

TIMOTHY GOMEZ

Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: Computer Science

The University of Texas Rio Grande Valley

May 2022

COMPUTATIONAL COMPLEXITY OF VERIFICATION IN TILE SELF-ASSEMBLY

A Thesis
by
TIMOTHY GOMEZ

COMMITTEE MEMBERS

Dr. Robert Schweller
Chair of Committee

Dr. Tim Wylie
Committee Member

Dr. Andres Figueroa
Committee Member

Dr. Bin Fu
Committee Member

May 2022

ABSTRACT

Gomez, Timothy, Computational Complexity of Verification in Tile Self-Assembly. Master of Science (MS), May, 2022, 44 pp., 4 tables, 13 figures, 54 references.

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 Thesis I will cover verification problems in different models of self-assembly leading to the proof that 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.

DEDICATION

To Uncle Charlie, thank you for inspiring me.

“If you love what you do you’ll never work a day in your life”

ACKNOWLEDGMENTS

I first and most importantly want to thank my parents for all their love and support. You both along with the rest of our family gave me everything I needed and more.

None of this would have been possible without my two advisors Robert Schweller and Tim Wylie. Schweller, thank you for introducing me to the world of research and self-assembly, and your consistent guidance. Wylie, thank you for helping me develop the tools I needed to excel. Thank you both for molding me into a proficient computer scientist.

I was not alone at alone at any part of this journey as I was surrounded by many other brilliant students at every step. David, we were a team for all of this and I can never thank you enough. Austin, thank you helping me while I was an undergraduate and showing me the template to be a great leader in the lab. I am so grateful for every member of the Algorithmic Self-Assembly Research group and Computer Science department for creating a supportive and friendly environment.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

Molecular Computing is a frontier of research where the natural properties and behaviors of molecules are studied as systems that perform computation. We see processes that we can algorithmically define, such as the patterns on the fur of animals, ants arranging themselves to larger structures, and even our cells performing actions based on their environment. It is of great interest for us to study these processes as models of computation to learn more about the natural world and it's relation to the theory of computing.

In my thesis I focus on Tile Self-Assembly models such as the 2-Handed Assembly Model, Staged Assembly Model, and Tile Automata. Most results involve either the computational complexity of verifying the behavior of systems in these models, or designing systems which implement algorithms. This is not meant to cover all topics in this area instead I focus on results leading to and around the main result of coNP-hardness of the Unique Assembly Verification in the 2HAM model in 2D with $\tau = 2$ along with some directions for future work in this area.

Shared Work Certain sections such were written with other coauthors. Many of the results covered here were joint work with coauthors David Caballero, Robert Schweller, and Tim Wylie [8–12] with some results stemming from initial ideas from Robert Schweller, Tim Wylie and Andrew Winslow [9, 10].

1.1 Models

Here I quickly review the main models explored and briefly review key related results. Other tile assembly models are viewed as generalizations are covered in the relevant section.

1.1.1 Abstract Tile Assembly Model

The first model of Tile Self-Assembly introduced was the aTAM in Erik Winfree's dissertation [50]. This models DNA structures as Wang Tiles which have already been studied. The first result in this model was showing that it is capable of Turing Computation.

Many of the problems I review were first explored in [1] such as the producibility and unique assembly verification problems. These problems were solvable in polynomial time in this model. Many generalizations of the problems and model have been studied such as what became known as the Unique Shape Verification problem which was introduced in [4]. This paper also introduced other generalizations such as the flexible glue model, multiple-tile model, and multiple temperature. This model is also the first place where Tile Assembly Computers [6, 35] and Covert Computation [17] are introduced.

1.1.2 2-Handed Assembly Model

When removing the restriction that growth starts from the seed we get the 2HAM where we define our producible assemblies recursively as any stable assembly we can build by combining two already producible assemblies. This model usually assumes starting with only single tiles (or size 1 assemblies). This model was introduced in [14] (although early versions had been proposed earlier [4, 24]) where many problems such as simulating the aTAM were studied. The main result for verification in this paper was coNP membership of UAV and coNP-Completeness when allowed one step into the 3rd dimension. This is the model generalized to three dimensions but tiles only place in $z = 0$ and $z = 1$. A notable technique used in this reduction were *cheat detection* assemblies which utilized both cooperative binding and geometric blocking to only allow attachment of two assemblies which represented the same variable assignments. This technique is used again many times in *test gadgets* a term used in [47] to show Π_2^p-Completeness of Unique Shape Verification

After many generalizations of UAV being shown to be hard [9, 11, 14, 46] it remained open for many years the complexity of UAV in the 2HAM in 2D with $\tau = \mathcal{O}(1)$. In [12] we prove that

this version of the problem is coNP-Complete via reduction from Monotone Planar 3SAT with Neighboring variable pairs [2]. This is an additional restriction that each clause with 3 variables has two that are adjacent.

1.1.3 Staged Assembly Model

Staged assembly uses a mix graph where each node or “bin" is a 2HAM system where the output of one pin are used as the input to another bin [24]. This model is much more powerful than the normal 2HAM only terminal assemblies are passed to the next bin. Verification was first studied in [47] showing up to Π_2^p-Completeness of UAV with 7 stages. Membership was shown to be in PSPACE with a conjectured PSPACE-Completeness.

The Unique Assembly verification problem in this model was shown to be PSPACE-Complete in [10, 47]. This reduction utilized covert computation which is possible using only 3 stages.

1.1.4 Tile Automata

The first models considered “passive" because tile only attach to each other and nothing is controlling process. The staged model can be considered active since the assembly process is being controlled the mix graph. Current developments in technology such as DNA Strand Displacement Circuits have inspired models of active self assembly. These models have tiles that can carry out slightly complex behavior such as in the Signal Tile model [42, 43]. The model we study is called Tile Automata [19]. This model adds the concept of state changes from Cellular Automata [**?**, 29, 31, 54] to the 2HAM. This model is very powerful as a computational model even at 1D in a lot of cases [8]. However many times it is of interest to limit this model in order to get closer modeling experiments.

1.2 Problems

One way to measure the power of a model is to study the problem of verifying the behavior of system. This can be viewed as a decision problem version of computing the output of a system.

Another way to think of this is debugging a designed system, we want to be sure that the desired assembly is actually produced by the system.

1.2.1 Producibility

The first problem we study is the producibility Problem. This just checks if there exists a build sequence to reach the target.

Definition 1.2.1 (Producibility Problem)**.** ***Input.*** *Tile Assembly system* Γ*, Assembly* A*.*

Output. *Yes/No: is* A *producible in* Γ*.*

1.2.2 Unique Assembly Verification

Verifying the output of an assembly system is very important. The unique assembly verification problem (UAV) asks whether an assembly is the unique terminal output. That is for every build sequence is the target produced. This can be seen as a computationally defined problem of debugging a self-assembly system when we know the output before hand. The definition of unique assembly varies slightly between models different cases need to be taken into account. We cover the definition of each model in the relevant chapters.

Definition 1.2.2 (Unique Assembly Verification Problem)**.** ***Input.*** *Tile Assembly system* Γ*, Assembly* A*.*

Output. *Yes/No: is* A *uniquely produced in* Γ*.*

1.2.3 Unique Shape Verification

Unique Shape Verification can be seen as a non-deterministic variant of UAV such as in [7] where we only care about the shape of the assembly.

Definition 1.2.3 (Unique Assembly Verification Problem)**.** [1] ***Input.*** *Tile Assembly system* Γ*, Shape* S*.*

Output. *Yes/No: is* S *uniquely produced in* Γ*.*

[1] Unique Shape is defined differently in [1].

1.2.4 Tile Assembly Computers and Covert Computation

Another type of problem in Tile Self-Assembly is constructing Tile Assembly Computers [6, 10, 17, 35]. Tile Assembly Computers are a non-uniform model of computation that consists of a tile set, and a way to encode the input and decode the output, with some bound on the power of the encoding methods. Initial work focused on the computing of specific functions rather than the powers of the computers as a whole. However it is a very nice frame work with which to study the power of self assembly systems.

Covert computation was introduced in [17] to address a feature many self assembling systems that compute functions have, they reveal the history. Usually the tile type that is placed at a location marks some step of the computation. However this tile can then be read revealing the input and entire computation history.

1.3 Experimental Motivation

Tile self assembly was motivated initially by [48] where plus shaped molecules of DNA are constructed that are stable and expose a sequence of base pairs on each side. This was taken further in [53] where the aTAM was used as one step in a chain from implementing circuits in the lab. More recently [21] designed and implemented DNA strand displacement circuits that are anchored on DNA origami to increase the speed of computation. This can give a sort of disconnected seed assembly which inspired the work of [23].

CHAPTER II

ABSTRACT TILE ASSEMBLY MODEL

The first model of self assembly introduced was the Abstract Tile Assembly Model (aTAM) [50]. In this Erik Winfree models the branch junction molecules of [48] using Wang Tiles and to describe the kinetic process of molecules attaching to a seed structure. This model served as the base for many other tile assembly models that generalize the model to have additional features or different kinds of growth.

Essentially, we have non-rotating square *tiles* that have a *glue* label on each edge. The tile with its labels is a *tile type* and a *tile set* is all the tile types. A glue function determines the strength of matching glue labels. An *assembly* is a single tile or a finite set of tiles that have combined via the glues. If the combined strength of the glue labels of a single attaching tile to an assembly is greater than or equal to the *temperature* τ, the tile may attach. A *producible* assembly is any assembly that might be achieved by beginning with the *seed*, the specified starting assembly, and attaching tiles. A producible assembly is further said to be *terminal* if no further tile attachment is possible. A tile system is said to *uniquely produce* an assembly A if all producible assemblies will eventually grow into A and A is terminal. A tile system is formally represented as an ordered triplet $\gamma = (T, s, \tau)$ representing the tile set, seed assembly, and temperature parameter of the system respectively.

2.1 Producibility

The first problem studied is the producibility problem with [1] which shows P membership in the aTAM. The Greedy Grow algorithm is a solution to the maximum produced subassembly problem which greedily attaches tiles to the assembly that match the target.

Problem	Temperature	Complexity	Reference	Theorem
Producibility	$\tau \geq 2$	$\mathcal{O}(\|A\|)$	[1]	2.1.2
UAV	$\tau \geq 2$	$\mathcal{O}(\|A\|^2 + \|A\|\|T\|)$	[1]	2.2.1
USV	$\tau \geq 2$	coNP-complete	[4]	2.3.1

Table 2.1: Known Results for verification problem in the aTAM. $|A|$ is the size of the target assembly, τ is the constant temperature of the system, and $|T|$ is the number of tile types in the system.

Definition 2.1.1 (maximum produced subassembly problem)**.** ***Input*** *aTAM system* Γ*, assembly A* ***Output*** *Largest assembly B such that B is a subassembly of A and B is producible in* Γ.

Since A is a subassembly of itself when A is producible the Greedy Grow Algorithm returns A. The algorithm runs in linear time by keeping a list of places where tiles may attach. However the run time presented in [1] does not take into account run time of calculating the binding strength but this is a very small factor (still takes polynomial time in the bits needed to store temp) increase especially since most system of interest have a constant temperature.

Theorem 2.1.2. *[1] The Producibility problem in the aTAM is in P with a* $\mathcal{O}(|A|)$ *time algorithm.*

The linear time greedy grow algorithm shows that simulating the growth of an aTAM system can be done on a computer quickly leading to multiple simulators having been developed.

2.1.1 Non-Cooperative Binding

A very well studied restriction of the aTAM is non-cooperative binding or $\tau = 1$. Cooperative binding is when a tile uses more than one strength $< \tau$ glue to bind. When $\tau = 1$ if a tile matches one glue it will attach. This limits the abilities of the model with much work being done to separate the restricted case from the general model [38–41] where most recently it was shown directed non-cooperative aTAM systems are not capable of Turing computation.

An interesting note though is that in this case Producibility is an easier problem to solve. Since determining if a tile t can be placed is just asking for a path in the assembly graph from the seed to t in the bond graph of the assembly. This might mean we have hope to solve Producibility and UAV for Non-Cooperative systems in a sub P complexity class such as NL. This also motivates

the question of whether both problems or maybe just UAV is P-Complete which I will cover more in depth in a later section.

2.2 Unique Assembly Verification

A polynomial time algorithm was shown in [1] using Greedy Grow from above as a subroutine. The intuition behind this algorithm is to check each location to make sure that only the desired tile can be placed in that location. If we want to check that the tile type t is the only tile type that can be placed at location l, we call greedy grow on $A - l$ to build the maximal subassembly that does not contain at tile at l. Once we do check can that only t can be placed there. Assume there exists a build sequence that builds an assembly B that is not A which differs at location l. Let t' be the tile at location l in B, assume t' is the first incorrect tile to appear as well. Since t' is the first correct tile to appear the assembly t' attaches to is a subassembly of A. This assembly will be built by greedy grow with no tile at l, then we will be able to see that t' attaches and the instance of UAV is false. If all locations only can place the target tile we know A is uniquely produced.

Theorem 2.2.1. *[1] The Unique Assembly Verification problem in the aTAM is in P.*

2.3 Unique Shape Verification

The previous problem takes in as input the system and a description of the assembly with all it's tile types. However what if we only care about the shape of an assembly. This problem is called the Unique Shape Verification problem and has been defined in two variants. Both are given a system Γ and a shape S. The first shown to be solvable in polynomial time [1] asks whether the system uniquely produces a single assembly that has the shape S. This problem asks a question for what is called Deterministic or Directed systems. In these systems only a single assembly is produced. However a generalization allows for multiple terminal assemblies as long as they have the same shape. This variant is called Non-Deterministic [7] or Unique-Shape Model [4]. In [7] the authors showed that finding the minimum tile set to build a shape in this model is Σ_2^{P}-complete compared to NP-Complete when only a single terminal assembly is allowed [1].

The first reduction I will cover also provides the base for many of the further reductions developed in Tile Self-Assembly [4]. This reduction works by encoding a *m* clause *n* variable SAT formula in a set of tiles as proposed first in [37]. Each column represent a variable and reach row a clause. Each row places tile to "evaluate" the clause to see if it's satisfied. The top left corner of this rectangle can only place if and only if there exists a satisfying assignment. This means that if there does exist a satisfying assignment the only thing that will be produced will be the rectangle missing a single tile.

Theorem 2.3.1. *[4] The Unique Shape Verification problem in the aTAM is coNP-Complete.*

This reduction fails in the case of UAV since the produced assembly encodes the satisfying assignment. This idea forms part of the inspiration of covert computation.

2.4 Tile Assembly Computers

A way to define how a tile assembly system can compute is by defining Tile Assembly Computers [6, 35].

Informally, a Tile Assembly Computer (TAC) for a function f consists of a set of tiles, along with a format for both input and output. The input format is a specification for how to build an input seed to the system that encodes the desired input bit-string for function f. We require that each bit of the input be mapped to one of two assemblies for the respective bit position: a sub-assembly representing "0", or a sub-assembly representing "1". The input seed for the entire string is the union of all these sub-assemblies. This seed, along with the tile set of the TAC, forms a tile system. The output of the computation is the final terminal assembly this system builds. To interpret what bit-string is represented by the output, a second *output* format specifies a pair of sub-assemblies for each bit. The bitstring represented by the union of these subassemblies within the constructed assembly is the output of the system.

Input/Output Templates. An n-bit input/output template over tile set T is a sequence of ordered pairs of assemblies over T: $A = (A_{0,0}, A_{0,1}), \ldots, (A_{n-1,0}, A_{n-1,1})$. For a given n-bit string

$b = b_0, \ldots, b_{n-1}$ and n-bit input/output template A, the *representation* of b with respect to A is the assembly $A(b) = \bigcup_i A_{i,b_i}$. A template is valid for a temperature τ if this union never contains overlaps for any choice of b, and is always τ-stable. An assembly $B \supseteq A(b)$, which contains $A(b)$ as a subassembly, is said to represent b as long as $A(d) \nsubseteq B$ for any $d \neq b$. We refer to the size of a template as the size of the largest assembly defined by the template.

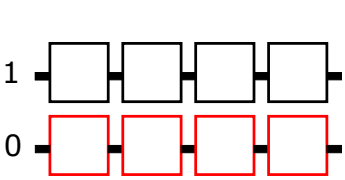
Function Computing Problem. A *tile assembly computer* (TAC) is an ordered quadruple $\Im = (T, I, O, \tau)$ where T is a tile set, I is an n-bit input template, and O is a k-bit output template. A TAC is said to compute function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^k$ if for any $b \in \mathbb{Z}_2^n$ and $c \in \mathbb{Z}_2^k$ such that $f(b) = c$, then the tile system $\Gamma_{\Im,b} = (T, I(b), \tau)$ uniquely assembles a set of assemblies which all represent c with respect to template O.

2.5 Covert Computation

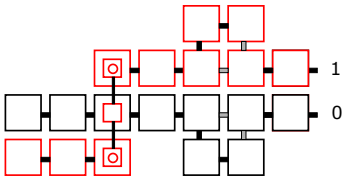
The function computing problem for TACs was originally introduced as a way to measure the ability of the model to compute specific functions such as addition, but we can view even the first Turing machine construction from [50] as a universal TAC for all computable functions. This construction however results in an output assembly which encodes the computation history. This property may be undesirable for computing functions where want the inputs to be secure, or irretrievable. To counter this [15] introduced the idea of Covert Computation which requires that the computation history is hidden. This is defined by required that if two inputs x and y that result in the same output, the seeds for each input must grow into the same assembly.

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

Outside of secure computing covert circuits were instrumental in the UAV reductions for the negative growth only aTAM [15] and the staged assembly model [10]. This shows a computational



(a) Dual Rail Wire



(b) Not Gate

complexity interest in the problem as well.

2.6 Negative Glues

When negative glues are allowed even the producibility problem becomes undecidable due to being able to detach tiles [28]. However we can still ask questions for what is called growth only systems [15]. Here the UAV problem is coNP-Complete with by introducing the idea of covert computation. This construction implemented NAND gates that "backfills" to hide it's inputs. The tile set can be constructed in polynomial time given a circuit serving as the reduction.

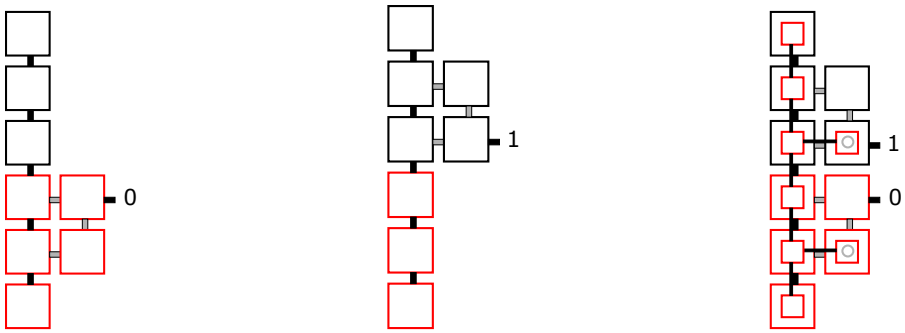
Theorem 2.6.1. *[15] The Unique Assembly Verification problem in the negative growth only aTAM is coNP-Complete.*

Theorem 2.6.2. *[15] The negative growth only aTAM is capable of covert computation.*

2.7 3D

The 3D aTAM generalizes our tiles to cubes with glues on up to 6 sides. Many papers focus on Just-Barely 3D constructions where tiles are placed at $z = 0$ and $z = 1$. The 3D aTAM with $\tau = 1$ has been show to be capable of simulating cooperative binding in [39].

A yet to be published result is the just barely 3D aTAM is capable of covert computation. The results works very similar to the construction in the negative aTAM by building a NAND gate with Dual Rail Logic. One difference though is the back filling occurs for all tiles at the end of the computation rather each gadget doing this individually. This result uses a polynomial assembly size.



(a) Input Assembly for 0 (b) Input Assembly for 1 (c) Backfilled Assembly

2.7.1 Wire

We will represent information being transferred using a wire. We construct a wire using two rows of tiles each representing a binary value of 0 or 1 as seen in Figure. During the computation only one row of each will be constructed. Once the computation is finished in order to hide the value the second row of tiles will be constructed.

Wires will be connected to gates which will perform steps of the computation. A simple example of a NOT gate can be seen in Figure 2.1b. This gate swaps the position of the rows of tiles. A row that represents a 0 will now be in the upper row and represent a 1. At the end of each gate is a diode gadget that was used in previous work. This prevents unwanted backfilling.

2.7.2 Input Assemblies

Our input assembly will consist of n 1×6 columns with two of four tiles attached on the right (Figures 2.2a and 2.2b). If the input bit is a 1 the top two tiles will be included and if 0 the bottom tiles. These tiles have enough attachment strength to be stable when both are present however since only tile may attach at a tile the other input is unable to grow.

Since we must have 1 final assembly regardless of the input or output once the computation is complete a bar will be assembled above in the second plane that will extend a tile above both sets of tiles and allow the unused inputs to construct.

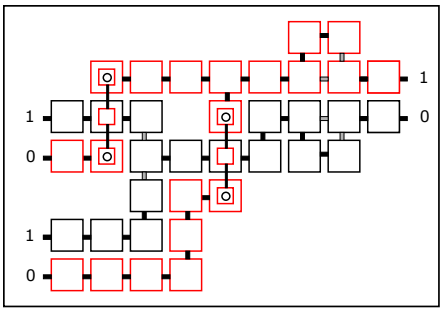


Figure 2.3: Full NAND gates

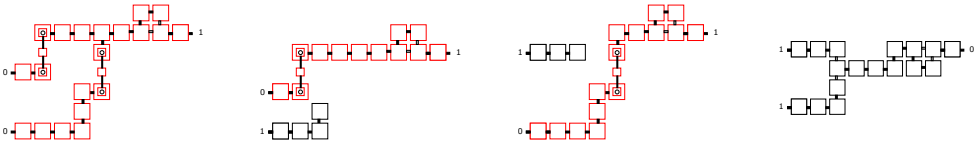


Figure 2.4: Possible partially build NAND gates.

2.7.3 NAND gate

We construct a NAND gate using the third dimension and cooperative binding. The full NAND gate can be seen in Figure 2.3. The tiles in black represent tiles that will be built from a 1 input and red represent 0. If either input to a NAND gate is 0 the output is always 1. If any red tile is placed the 1 output of the gate will be built. If both inputs are 1 the 0 output can be constructed using cooperative binding. This can be seen in Figure 2.4.

One thing to note in the case of one output being 0 and the other 1 is that the red tiles will be placed along the other wire. However this will not cause any issues since it can only build back up to the output of the previous gate due to the diode gadget.

2.7.4 Back Filling and Target Assemblies

In order to perform covert computation there must exist a unique assembly for each output. The grey tile at the end of the circuit will be one of two flag tiles that will denote the output of the circuit. Once this tile is placed a row of tiles will be built back towards the inpu. Once the input assembly is reached the tiles above the input will be placed allowing for the input assemblies to be filled in. This will cause the whole circuit to be filled out hiding the orginal input and computation

history. It is still open whether covert is possible in the 2D aTAM however it has been conjectured it is not possible in 2D with only positive glues and a polynomial assembly side.

Theorem 2.7.1. *The 3D aTAM is capable of covert computation.*

2.7.5 P-Completeness

We note the covert construction in 3D does not serve as a NP-hard reduction like the negative aTAM. This is because the seed assemblies to encode the input are not producible at $\tau = 2$ (the seed is stable however it not producible by attaching a single tile). While this does not give NP-hard we can use it for a P-hard reduction. Given a Boolean circuit we can construct the tile set of the covert TAC using logarithmic in the input length space. This means we can perform a logspace reduction from the Circuit Value Problem which is P-Complete to both the Producibility and the UAV problem. Since the Produciblity algorithm from [1] also works in 3D this means that both problems are P-Complete.

Theorem 2.7.2. *Producibility in the 3D aTAM is P-Complete with* $\tau = 2$.

This motivates further investigation into covert computation and it's affect on the complexity of verification problems in the 2D aTAM. While the aTAM can carry out computation it leaves the history, which was the motivation for covert. This causes issues when trying to perform reductions lower than P as we are not able to compute the target assembly. If we cannot perform covert computation perhaps it is because these problems are solvable in easier than P. It is of note that while Monotone CVP and Planar CVP are both P-Complete, Monotone Planar CVP is in NC^3. Monotonicity and Planar are two problems that arise often when constructing circuits in the aTAM so perhaps a similar upper bound can be shown for verification? Or maybe the aTAM has other properties which can overcome these issues.

2.8 Beyond Tiles

Many models have been explored with larger or different shaped pieces instead of tiles. In [30] these larger pieces are defined as arbitrary shaped tiles or polyominoes. That we still only

attach one tile at a time, but a tile occupies multiple positions. This model allows for universal computation with non-cooperative binding, by utilizing geometric blocking. A polyomino tile cannot attach if it overlaps with an already placed tile.The dupleTAM uses size 1 and 2 polyominoes [34] and shows that it is capable of universal computation. The aTAM has also been generalized with arbitrary polygon tiles [25] where it was shown a single tile type is can simulate a Turing machine encoded in the seed.

The producibility problem becomes NP-Complete with constant sized prebuilt assemblies as the result from [11] extends to this model. However this model is better defined as a restricted version of the 2HAM so I cover it in the next chapter.

CHAPTER III

2 HANDED ASSEMBLY MODEL

The 2-handed assembly model (2HAM) has been shown to be more powerful than the aTAM in its program-size efficiency for finite shapes [13], and its running-time efficiency for the self-assembly of finite shapes [22]. Here I cover an overview of verification results in the 2-handed assembly model and closely related generalization.

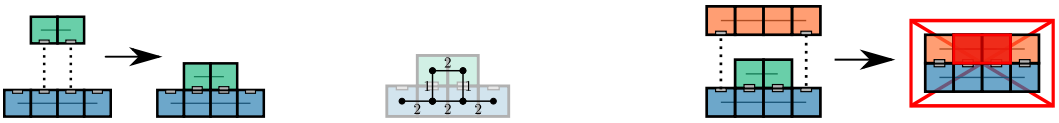
3.1 Definitions

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* $\text{str}(g_1, g_2)$.

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 $\text{dom}(\tilde{A})$ 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 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



(a) Cooperative Binding (b) Bond Graph (c) Geometric Blocking

Figure 3.1: (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.

least τ. This means that the sum of the glue strengths along each cut is greater or equal to τ. An example bond graph for a small assembly is shown in Figure 3.1b.

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 exists 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 3.1a. 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 3.1c.

Problem	Temperature	Complexity	Reference	Theorem
Producibility	1	$\mathcal{O}(\|A\|\|T\|)$	[27]	Thm. 3.2.2
Producibility	$\tau \geq 2$	$\mathcal{O}(\|A\|\log^2\|A\|)$	[27]	Thm. 3.2.1
UAV	1	$\mathcal{O}(\|A\|\|T\|\log\|T\|)$	[27]	Thm. 3.2.2
UAV	$\tau \geq 2$	coNP-complete	[12]	Thm. 3.3.4
USV	$\tau \geq 2$	coNP^{NP}-Complete	[47]	Thm. 3.4.2

Table 3.1: Known Results for verification problem in the 2HAM 2D. $|A|$ is the size of the target assembly, τ is the constant temperature of the system, and $|T|$ is the number of tile types in the system.

Two-handed Assembly. A two-handed assembly system (2HAM) is an ordered tuple $\Gamma = (T, \tau)$ where T is a set of single tile assemblies 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 $C \in P'_\Gamma$.

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.

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-tile assemblies 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.

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 the following are true: 1) $P_\Gamma = \{A\}$, and 2) For all $B \in P'_\Gamma$, $|B| \leq |A|$.

3.2 Producibility

The problem of determining whether or not a given assembly is producible in a given system is solvable in polynomial time [27]. At a high level this algorithm works by starting with a graph of single tile assemblies that are connected to neighboring tiles in the target assembly then repeatedly making possible attachments. The key ideas behind them algorithm are that attaching a tile can never prevent the attachment of a later tile in the build sequence, so the build sequence doesn't matter. This means we can again greedily combine assemblies. [27] goes beyond this method to show how the choice of data structure can increase the run time.

Theorem 3.2.1. *[27] The Producibility problem in the 2HAM is in* $\mathcal{O}(|A|\log^2|A|)$.

3.2.1 Temperature-1

At temperature-1, where we only have non-cooperative binding, [27] a faster algorithm is shown based on a fast algorithm for aTAM. Without cooperative binding the 2HAM and aTAM are much more similar. It is shown that this method can also be used to solve UAV as we just need to treat each tile as the possible seed.

Theorem 3.2.2. *[27] The Producibility and UAV problem in the 2HAM is in* $\mathcal{O}(|A|^2|T|+|A||T|^2)$ *and* $\mathcal{O}(|A|^2|T|^2+|A||T|^3)$.

3.3 Unique Assembly Verification

Membership in coNP for the UAV problem was shown in [14] relying on the produciblity proof from [27]. They also showed the problem was coNP-Complete in 3D by using a reduction that utilized geometric blocking and one of the first use of what would later be called *test assemblies*, an important technique for proving hardness. The techniques of geometric blocking were also used and developed in the staged assembly model as well.

3.3.1 Rogue Assemblies

Definition 3.3.1 (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 in [12], about what we call *rogue assemblies* which helps with proof of reduction This lemma states that if the instance of UAV is false and all the tiles in Γ are used to build A, then we may find a rogue assembly by only checking combinable subassemblies of A.

Lemma 3.3.2. *For an instance of UAV* (Γ, A) *that is false, either there exist two 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 single tile assembly* $t \in P'_\Gamma$ *that is not a subassembly of* A*.*

3.3.2 UAV is coNP-hard

While the aTAM has a polynomial time solution to the UAV problem [1], allowing for the production of efficient simulators [32, 44, 51], the complexity of UAV in the 2HAM has remained a long-standing open problem in the field. The 2HAM appeared formally in 2013 [14], but was essentially defined in staged self-assembly [24] (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 (which I cover more in depth later in this chapter), 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 [26, 45, 52, 53] for surveys and applications of self-assembly theory. Here I summarize the proof of NP-hardness of the canonical form of the UAV problem, in 2 Dimensions and with $\tau = 2$. A full version of this result can be found in [12]. Monotone Planar 3-SAT with Neighboring Variable Pairs was recently shown to be NP-hard in [2]. 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 [36]. 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 3.2a.

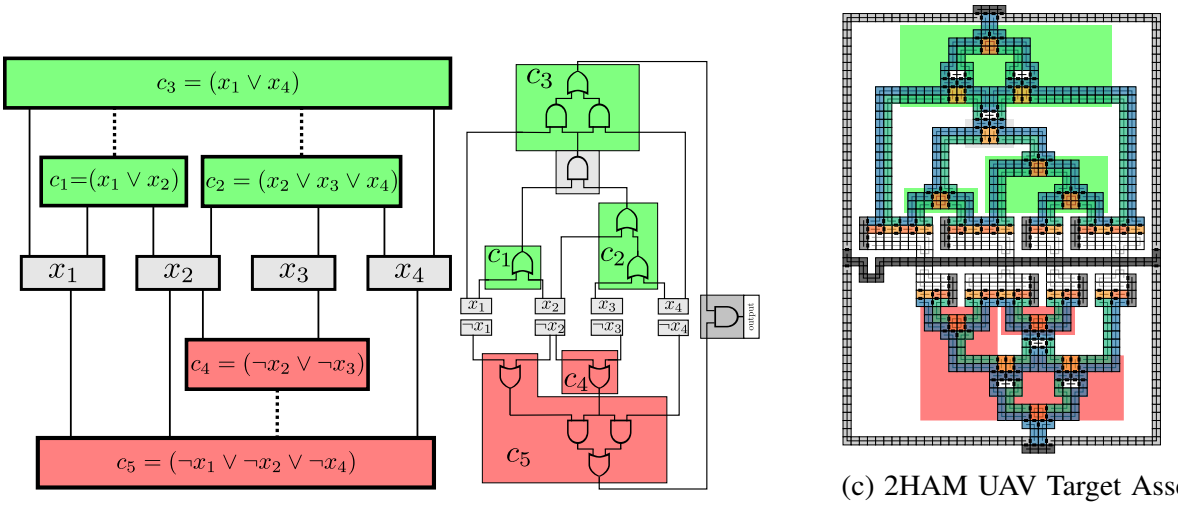
Problem 3.3.3 (Monotone Planar 3-SAT with Neighboring Variable Pairs (MP-3SAT-NVP))**.** ***Input:*** *Boolean formula $\phi = C_1 \wedge \cdots \wedge C_m$ in 3-CNF form where each clause only contains positive or negated literals from $X = \{x_1, \ldots, x_n\}$. Further, any clause of ϕ with 3 variables 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?*

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. 3.2a and 3.2c. 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 3.2a, 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]

Theorem 3.3.4. *[12] The Unique Assembly Verification problem in the 2HAM is in coNP-Complete at $\tau \geq 2$.*

[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 clauses (one for the positive and negative sides).



(a) MP-3SAT-NVP Instance (b) Target Assembly Circuit (c) 2HAM UAV Target Assembly

Figure 3.2: (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.

3.3.3 Tree Shaped Assemblies

If the bond graph of the target assembly is a tree then UAV is solvable in polynomial time even when the temperature can be specified in the input [12]. While this is not a very natural case of the UAV problem as the only time rogue assemblies may form is when there are exposed glues that are never used on the target assembly. However this algorithms serves as an interesting separation between cooperative and non-cooperative binding not solely based on the temperature of the system.

In order for there to be cooperative binding a loop must be including in the bond graph. Our algorithm takes advantage of this by using the polynomial time algorithm to verify that the tile was the unique assembly producible using non-cooperative attachments. Then we show a dynamic programming method the first cooperative binding that can take place.

3.4 Unique Shape Verification

A key result in the 2HAM was that the unique shape verification problem is coNP$^{\text{NP}}$-Complete [47]. Using the idea of geometric blocking utilized previously in [14] this format was followed in many further reductions. The reduction from ∀∃SAT utilizes a technique of non-deterministically constructing *computation* assemblies and *test* assemblies. These two types of assemblies can combine if they represent the same assignments. We can view these assemblies as each evaluating a quantifier. An overview of the reduction can be seen in Figure 3.3.

Definition 3.4.1 (∀∃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 $\phi(X,Y) = 1$?*

Computation Assembly A computation assembly is built for every assignment to both sets of variables as in [4, 37] and exposes a glue that marks whether the formula was satisfied. The reduction adds an additional geometric encoding of the assignment to variables in X that is also exposed. This can be thought of as the there exists part of the reduction as there exists an assembly with a satisfied glue encoding an assignment to X if there exists a Y where the formula is true.

Test Assembly Test assemblies are built for all assignments to X which encode the assignment . A test assembly may attach to a computation assembly that outputs true and has matching variables assignments. The variables are encoded in the geometry of the shape. A test assembly is terminal if for that assignment to X there does not exist a computation assembly that satisfies the assignment. The test assemblies work as the for all part of the reduction as the target shape is uniquely produced only if for all assignments to X there exists some Y that allowed a matching computation assembly to build.

Theorem 3.4.2. *[47] The Unique Shape Verification problem in the 2HAM is in coNP^{NP}-Complete at $\tau \geq 2$.*

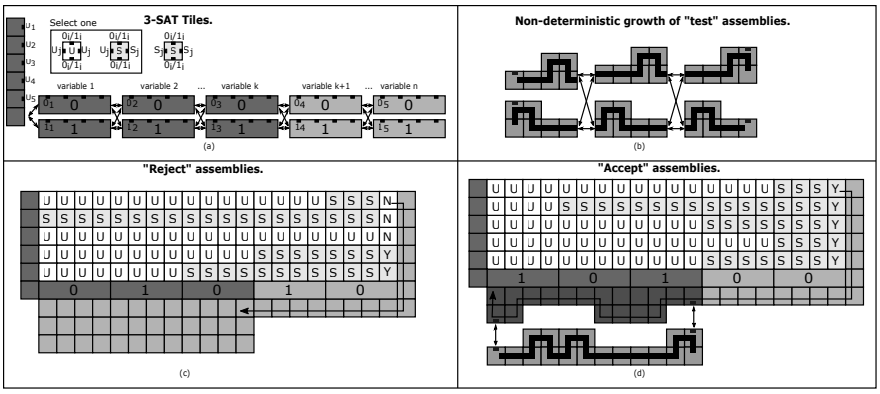


Figure 3.3: Overview of 2HAM USV Reduction. Top right the frame builds with an assignment to the variables. Top left test assemblies build for all assignments to X. Bottom left assemblies that don't satisfy the formula go to the target shape. Bottom right assignments that do satisfy the formula encode their assignment to the X variables in their geometry. Test assemblies may attach to matching computation assemblies.

3.5 Generalizations

2HAM has been generalized in many ways providing a wide range of models. Here are some notable results that do not have enough for another section but still have results of note for verification problems.

3.5.1 High Temperature

If the temperature is allowed to be specified in the input [46] shows coNP-Completeness of UAV via a reduction from Hamiltonian Path. The temperature used in the reductions is equal to the number of nodes in the graph. The construction works by creating a frame that represents the outline of the graph. Then possible walks through the graph are encoded in the producible assemblies. An assembly has a number of glues exposed equal to the number of nodes in the path. If each vertex was visited then this assembly can fit into the frame forming a rogue assembly. This is a simple generalization as it doesn't change any definitions of the model, however encoding information in the temperature has assisted in more efficient construction of general shapes [20].

3.5.2 Multiple Tile Model

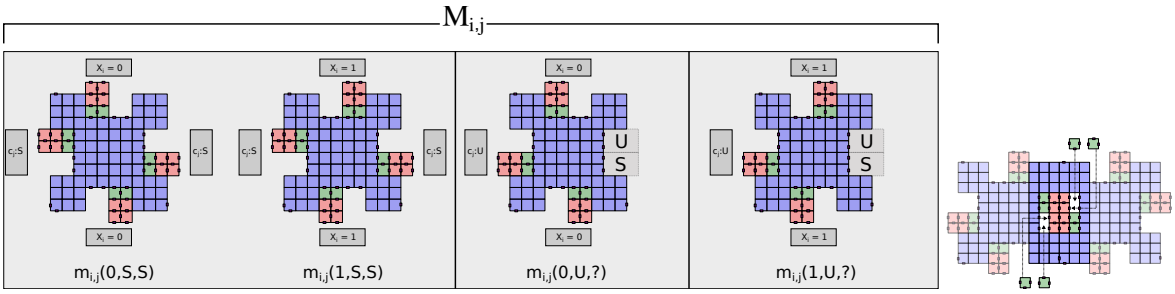
A middle ground between the aTAM and the 2HAM is the Multiple Tile or q-tile model where assemblies up to size q are allowed to grow in a fashion to 2HAM to form a set of supertiles. Growth starts from a seed and any assembly from our set of super tiles may attach. Under this definition UAV in the Multiple Tile Model is hard with a polynomial q using the reduction from above. Since $q = 1$ is the aTAM this problem is solvable in polynomial time. Is UAV with a constant q always solvable in polynomial time?

3.5.3 Prebuilt Assemblies

When larger than single tiles are allowed Producibility and UAV increases in hardness to NP-Complete and coNP$^{\text{NP}}$-Complete respectively. The Producibility reduction works by evaluating a 3SAT formula in a rectangle similar to [4, 37] using macroblocks where the "edge labels" are encoded in geometry as well rather than glues. The main challenge for designing this reduction is there must exist a single assembly that is produced regardless of the satisfying assignment. We do this by including a certain subset of the tiles which will fill any spaces left between macroblocks as shown in Figure 3.4. We use the same method of geometry encoding to build test assemblies to implement a similar reduction for UAV as the reduction in [47] for USV.

3.5.4 Multiple hands

When allowing more than 2 assemblies to come together in a single step we get the multiple hand model. One proven strengthen of this model is its ability to build infinite fractal patterns [18,33]. We study verification problems in [9] with UAV being coNEXP-Complete when the number of hands can be exponential in the input size. Produciblity stays in P regards of the encoding of the hands.



(a) (b)

Figure 3.4: (a) Possible Macroblocks that make up $\mathscr{M}_{i,j}$. 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, $m_{i,j}(1,U,S) \in \mathscr{M}_{i,j}$, otherwise $m_{i,j}(1,U,U) \in \mathscr{M}_{i,j}$. If the negative literal $\bar{x_i}$ is in c_j, $m_{i,j}(0,U,S) \in \mathscr{M}_{i,j}$, otherwise $m_{i,j}(0,U,U) \in \mathscr{M}_{i,j}$. (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.

CHAPTER IV

STAGED ASSEMBLY MODEL

Staged Assembly Model generalizes the 2HAM to have multiple bins, connected in a mixed graph. A mix graph is a directed graph divided into s stages. Each bin in stage 1 starts with a subset of the tiles and acts a 2HAM system. The terminals then mixed according the edges of the mix graph into bins in stage 2 which functions as a 2HAM system with prebuilt assemblies. The process is then repeated.

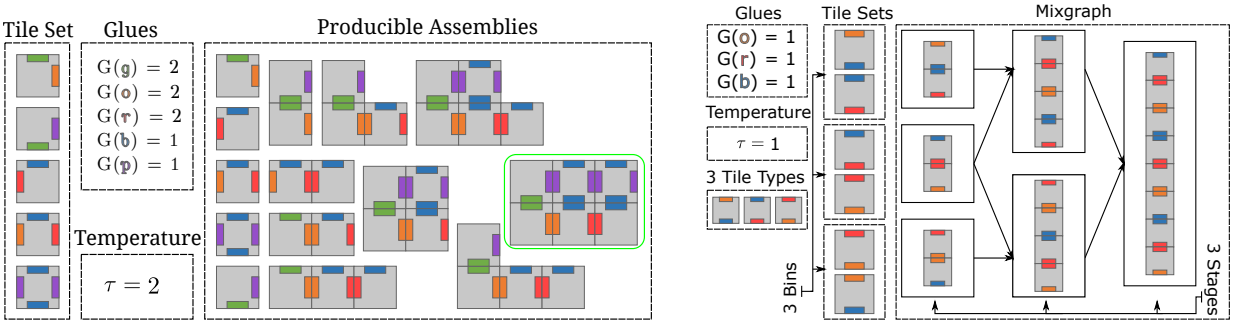
Staged assembly systems. An *r-stage b-bin mix graph* $M_{r,b}$ is an acyclic r-partite digraph consisting of rb vertices $m_{i,j}$ for $1 \leq i \leq r$ and $1 \leq j \leq b$, and edges of the form $(m_{i,j}, m_{i+1,j'})$ for some i, j, j'. A *staged assembly system* is a 3-tuple $\langle M_{r,b}, \{T_1,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

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

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

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

[1] The original staged model [24] only considered $\mathcal{O}(1)$ distinct tile types, and thus for simplicity allowed tiles to be added at any stage. Since our systems may have super-constant tile complexity, we restrict tiles to only be added at the initial stage.



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

Figure 4.1: (a) A 2HAM example that uniquely builds a 2×3 rectangle. The top 4 tiles in the tile set all combine with strength-2 glues building the 'L' shape. The tile with blue and purple glues needs two tiles to cooperatively bind to the assembly with strength 2. All possible producibles are shown with the terminal assembly highlighted. (b) A simple staged self-assembly example. The system has 3 bins and 3 stages, as shown in the mixgraph. There are three tile types in our system that we assign to bins as desired. From each stage only the terminal assemblies are added to the next stage. The result of this system is the assembly shown in the bin in stage 3.

stage.[2] We say this set of output assemblies is *uniquely produced* if each bin in the staged system uniquely produces its respective set of terminal assemblies.

4.1 Covert Circuits

Tile assembly computers were defined in [15, 35] with respect to the aTAM. We provide formal definitions of both Tile Assembly Computers and Covert Computation with respect to the Staged Self-Assembly model.

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

[2]This is a slight modification of the original staged model [24] in that the final stage may have multiple bins. However, all of our results apply to both variants of the model.

interpret what bit-string is represented by the assembly, a second *output* format specifies a pair of sub-assemblies and locations for each bit. An assembly that represents a bitstring is created by the union of each sub-assembly represented by each bit.

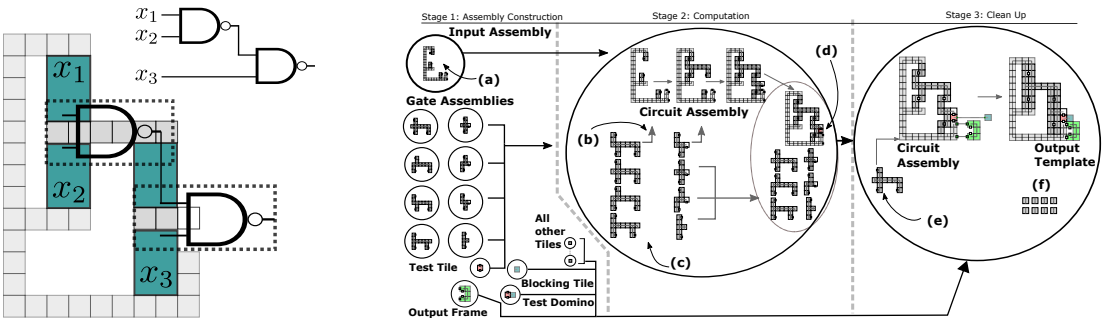
For a STAC to *covertly* compute f, the STAC must compute f and produce a unique assembly for each possible output of f. Thus, for all x such that $f(x) = y$, a covert STAC that computes f produces the same output assembly representing output y for each possible input x, making it impossible to determine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

Staged assembly is able to perform covert computation with 3 stages [10]. Figure 4.2b shows a basic overview of the mixgraph used for the covert computation implementation. The method requires three stages with a polynomial number of mixing bins.

- In the first stage, we assemble the components needed to perform the computation. These include an *Input Assembly*, which encodes the input to the function, *Gate Assemblies*, which act as individual gates and perform the computation via their attachment rules and geometry, and additional assemblies which are used to help "clean up" our circuit and covertly get the output.

- In stage two, the input assembly and gate assemblies are added to a single bin along with a test tile. The gate assemblies will begin to attach to the input assembly creating a *Circuit Assembly*. Once the computation is complete, the test tile can attach to the circuit assembly if and only if the output is true. The circuit assembly is terminal in this bin and will be passed to the final stage.

- The final stage adds additional assemblies to the bin along with most of the tile set as single tiles (not shown in figure). The additional assemblies read the output of the circuit and it grows into one of the output templates. The *Output Frame* searches for the test tile representing the output of the circuit. The single tiles fill in any spaces left in the circuit assembly that would show the computation history, thereby turning the assembly into the output template. This

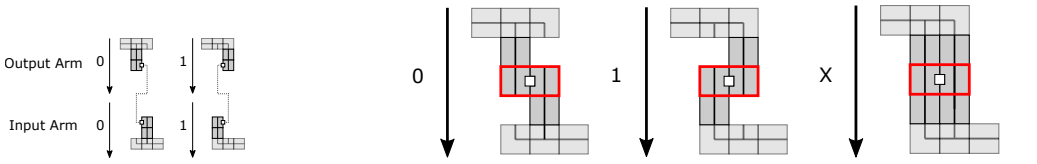


(a) (b)

Figure 4.2: (a) Simple 3-input logic circuit using 2 NAND gates, and the high-level abstraction of the circuit assembly showing the input variables and gates highlighted as blocks. Blue blocks are the sections of the assemblies we call *Arms* that function as wires in the systems. (b) (1) Our input assembly and gate assemblies are constructed in separate bins. (2) Gate assemblies attach to the input assembly forming a circuit assembly. (3) Unused gates are terminal in the second stage. (4) This circuit evaluates to true, so the test tile will be able to attach. (5) Gate assemblies in this stage grow into a circuit using single tiles. (6) Single tiles fill in open spots in the circuit assembly to hide the history. The additional assemblies are used to reach the output template.

requires a linear number of additional bins in the first and second stage to store these single tiles while mixing takes place in other bins.

For our circuit assembly we implement Planar Logic Circuits with only NAND gates. An example circuit and an assembly showing how the gates are laid out are shown in Figure 4.2a. Wires are represented by 2×3 blocks of tiles shown in blue in the image. Input and Gate assemblies contain a subset of the tiles in each block we call *arms* which represent the values being passed along the wires as in Figure 4.3a. The input assembly is a comb-like structure that is designed so that each input bit reaches the gate it is used at. For each NAND gate in the circuit we have 4 different assemblies, one for each possible input to the gate. A gate assembly can cooperatively bind to the input assembly if the variable values match. The gate assembly has a third arm that represents the output. This allows the next gate assembly to attach, which continues propagating until the computation is done and the circuit assembly is complete. Again as in the 2HAM with prebuilt assemblies there exists a gap that still displays the bit that was passed. We solve this in the same way by filling it in with single tiles Figure 4.3c, but it must be done in the third stage.



(a) (b) (c)

Figure 4.3: (a) Information being passed along a wire is represented by the position of a domino called an arm. Output arms represent a signal of "1" of "0" by being in the left or right position, respectively. Input arms read bit values and have complimentary arm placement to allow for attachment. (b) Information is passed by attachment. Another assembly may attach if the arms have matching glues (they represent the same wire) and they have complementary arms (represent the same bit value). (c) In the final stage we add additional tiles to hide the information that was passed along a wire.

Stages	**Membership**		**Hardness**	
1 (2HAM)	coNP	In [14]	coNP-complete	In [12]
2	Π_3^p	**Thm. 4.2.4**	coNP-hard	In [12]
3	Π_4^p	**Thm. 4.2.4**	Π_2^p**-hard**	**Thm. 4.2.1**
$n > 3$	Π_{n+1}^p	**Thm. 4.2.4**	Π_{n-1}^p**-hard**	**Thm. 4.2.3**
General	PSPACE	In. [47]	**PSPACE-complete**	**Thm. 4.2.2**

Table 4.1: Complexities of Unique Assembly Verification in the Staged Assembly Model with respect to the number of stages n. Our results are in bold.

4.2 Unique Assembly Verification Problem

In the staged assembly model, initial investigation in [47] showed coNP-hardness using four stages and Π_2^p-hardness for seven stages. They also showed membership in PSPACE with a conjecture of PSPACE-completeness.

In this [10], we use the covert construction from above to show UAV is PSPACE-complete in staged self-assembly, resolving the open problem from [47]. Along the way, we improve on some results from [47]: we show that UAV is Π_2^p-hard with just three stages, improving on the previous hardness result requiring seven stages. We then generalize this result to show that for n stages, UAV is Π_{n-1}^p-hard, but yields a Π_{n+1}^p algorithm, leaving only a gap of two in levels between membership and hardness for this problem. An overview of our results and known results related to UAV is shown in Table 4.1.

Three Stages Given an instance of ∀∃SAT, we create a 3-stage temperature-2 staged system that uniquely produces a target assembly iff the given instance of ∀∃SAT is true. The reduction uses the same high-level idea as [47] and [8]. The process begins with the construction of an assembly for every input to the ∀∃SAT formula. Circuit assemblies build from these inputs and are flagged as true or false, while encoding a partial assignment through their geometry. Separate "test" assemblies are constructed that also encode a partial assignment to the same variables, which attach to true circuit assemblies with matching assignments. The systems uniquely assembles a target assembly if for all test assemblies there exists a compatible true circuit assembly for it to attach to.

Theorem 4.2.1. *[10] UAV in the Staged Assembly Model with three stages is* Π_2^p*-hard with* $\tau = 2$.

General UAV We utilize the same technique used in above which reduced from ∀∃SAT, a special case of $TQBF$ limited to only 2 quantifiers, but adapt the technique to work with a QBF with n quantifiers $\forall x_1 \exists x_2 \ldots \forall x_n(\phi(x_1,\ldots,x_n) = 1)$. In the 3rd stage, instead of adding in single tiles to "clean up", we add in a second set of test assemblies that represent an assignment with one less variable in the next stage and are complementary in their geometry. These new test assemblies then attach to previous test assemblies that were terminal in the previous stage with matching partial assignments. This process computes an additional quantifier. We can then repeat this process of adding in complementary sets of test assemblies for the number of quantifiers required. In the final stage, if a test assembly from the final set couldn't find a complementary test assembly to attach to, the instance of TQBF is false, and that test assembly is prevented from growing to the target assembly. This allows the truth of instance of staged UAV to correspond to the truth of the QBF.

Theorem 4.2.2. *[10] Unique Assembly Verification in the Staged Assembly Model is PSPACE-complete with* $\tau = 2$.

Specifically we may also bound the specific level of PH that n stage UAV is hard for.

Theorem 4.2.3. *[10] For all* $n > 1$*, UAV in the Staged Assembly Model with n stages is* Π_{n-1}^p*-hard with* $\tau = 2$.

4.2.1 Membership

Improved Membership bounds were shown in [10] lowering the gap between membership and hardness to a constant number of levels of the Polynomial Hierarchy. We do so using a similar method of using 3 promise problems asking about if an assembly is producible or terminal in a bin, and asking the max assembly size in a bin. By utilizing oracles to promise problems rather than the more general algorithms allowed for a stricter membership algorithm lowering the member to only a additive factor above the number of stages.

Theorem 4.2.4. *The n-stage Unique Assembly Verification problem in the staged assembly model is in* Π^p_{n+1}.

CHAPTER V

TILE AUTOMATA

The Tile Automata model was introduced in [19] merging ideas from Cellular Automata and Tile Self-Assembly. The authors showed that *freezing* tile automata (where a tile cannot repeat states) is capable of simulating non-freezing systems. This powerful model has also been shown to be capable of simulating models of programmable matter [5]. A model motivated by real-world implementations, the Signal-passing Tile Assembly Model, is able to simulate Tile Automata [16] meaning results shown in the TA model carry over to STAM at scale. Ameobots is a model of programmable matter where the agents are allowed to move and expand/contract. Tile Automata has also been shown to be capable of simulating this model [5].

In [8] we investigated Tile Automata and it's ability to do computation. These results were based around the ability for Tile Automata to simulate a Turing machine even at 1D. We can divide the results into two areas. The first is building $1 \times n$ lines. While I do no cover those in depth for non-freezing systems it was shown that it is possible to build busy beaver length lines based on simulating a Turing machine.

5.1 General

The ability for tiles to detach usually gives undecidable results for problems like UAV. This is shown in [8]. Even when we add the freezing restriction we can still simulate a Turing machine by replacing tiles using the method from [19]. However this uses macroblocks of scale 5×5 so in an effort to minimize the use of the second dimension we show that we can do this even with assemblies of only height-2.

Transition Rules	Freezing	1D Result	2D Result	Theorem
Affinity Strengthening	Freezing	$P_{\|\|}^{NP}$-Complete	$coNP^{NP}$-Complete	Thms. 5.2.5, 5.2.2
Affinity Strengthening	Non-freezing	PSPACE-Complete	PSPACE-Complete	Thm. 5.1.3
General	Freezing	Open	Undecidable	Thm. 5.1.2
General	Non-freezing	Undecidable	Undecidable	Thm. 5.1.1

Table 5.1: Results for the Unique Assembly Verification in Tile Automata. **Transition Rules** describes the types of transition rules allowed in the system. In Affinity Strengthening Systems all transition rules increase affinity so no detachment may occur. **Freezing** indicates whether the system is freezing where tiles cannot repeat states. **Result 1D** is the complexity of UAV in one dimension and **Result 2D** is the complexity of UAV in two dimensions. **Theorem** is where these results can be found.

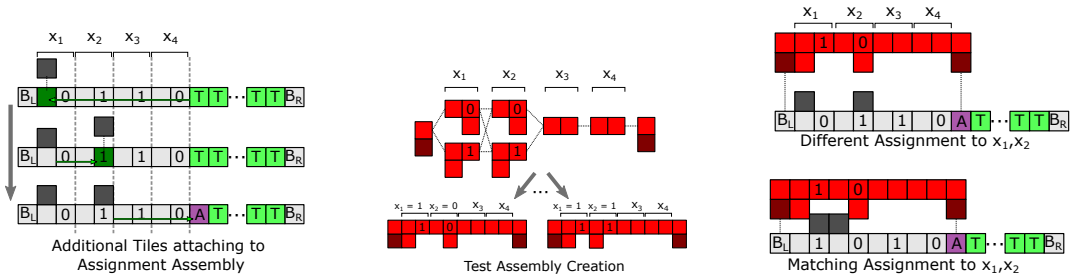
Theorem 5.1.1. *[8] Tile Automata Unique Assembly Verification is undecidable in one dimension.*

Theorem 5.1.2. *[8] Freezing Tile Automata Unique Assembly Verification is undecidable even when all assemblies are of max height-*2*.*

However there is still no result for only height 1 Freezing Tile Automata. We cannot perform the same tile replacement as in 2D as removing a tile in the middle of the assembly will result in it falling off. Currently the only Turing machine in Freezing without detachment can only simulate bounded time Turing machines as each time the head moves over a cell it uses up a state. Thus you cannot walk arbitrarily back and forth. Is UAV in 1D freezing Tile Automata decidable? Or does there exist some other computational model that may be simulated.

5.1.1 Affinity Strengthening

With Affinity Strengthening Tile Automata we now have an upper size limit on the assemblies need to check as if we ever build something larger than our target we can reject as with many of the previous models. We have shown PSPACE-completeness of UAV under this restriction. As



(a) (b) (c)

Figure 5.1: Example of ∀∃SAT over 4 variables with $k = 2$. (a) Transition rules pass a signal to the left that transitions the tile next to the state representing x_1, allowing a tile to attach. This allows more transitions that allow a tile to attach over the state representing x_2 (since it is $= 1$). Finally, the leftmost T is transitioned to state A. (b) Four test assemblies are built, each representing an assignment to x_1, x_2. A test assembly can attach to an assignment assembly with the A state if they encode the same assignment to x_1, x_2.

we can simulate a Turing machine under this restriction, however we cannot detach tiles to make them fall apart. Thus were left with an assembly that is the size of the space we used.

Theorem 5.1.3. *[8] The Unique Assembly Verification problem in Affinity Strengthening Tile Automata is PSPACE-complete.*

5.2 Freezing Affinity Strengthening

Taking the idea of computation and test assemblies to the lowest height possible creating a reduction with a target assembly of height 3. We can simulate a non-deterministic polynomial time Turing machine in one dimension which functions as the computation assembly. If it satisfies the formula tiles attach building geometry that encodes the assignment to X for a coNP$^{\text{NP}}$-hard reduction for UAV.

Definition 5.2.1 (∀∃3SAT)**.** *Given a 3SAT formula $\phi(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$, is it true that for every assignment to variables $x_1, \ldots, x_k$, there exists an assignment to $x_{k+1}, \ldots, x_n$ such that $\phi(x_1, \ldots, x_n)$ is satisfied?*

Theorem 5.2.2. *[8] Unique Assembly Verification in height-3 freezing Affinity Strengthening Tile Automata is coNP^{NP}-complete.*

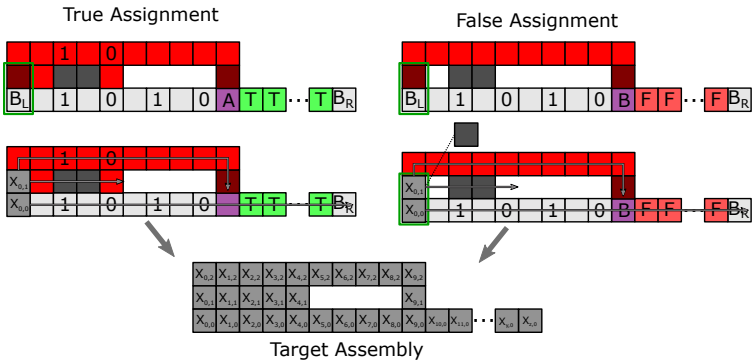


Figure 5.2: An overview of how all assignment assemblies transition to the target assembly. Assignment assemblies marked true can grow to the target assembly by attaching to a test assembly. Those flagged false can attach to "blank" test assemblies, and the filler tile fills in the spots missing relative to the target assembly.

5.2.1 One Dimensional

The case of 1-Dimensional Freezing Affinity Strengthening Tile Automata revealed a interesting result. It is solvable in $P_{||}^{NP}$ which the problems solvable in Polynomial time with access to a NP oracle, however only a polynomial number of queries that must all be made at the same time.

Definition 5.2.3 ($P_{||}^{NP}$)**.** *Class of problems solvable by a deterministic Turing machine in polynomial time that is allowed a single query to a polynomial number of parallel NP oracles.*

Definition 5.2.4 (Max True 3SAT Equality)**.** *For a 3-CNF formula F, let max-$\mathbb{1}(F)$ denote the maximum number of variables set to true in a satisfying assignment to F. Given two 3-CNF formulas F_1 and F_2, is it true that max-$\mathbb{1}(F_1) =$ max-$\mathbb{1}(F_2)$?*

Max True 3SAT Equality is complete for the class $P_{||}^{NP}$ [49]. Given an instance of this problem, the reduction provides an instance of 1D freezing ASTA UAV.

Overview At a high level, given an instance of Max True 3SAT Equality (F_1, F_2), we utilize the same 1D Turing machine simulation to represent each assignment/formula pair with an assembly, and flag these assemblies as True or False. We modify the system to allow it to count the number of

ones that are contained in the true assemblies, and add a "count" state to the left/right edge of these assemblies that reflects this number. To reach the target assembly, an assembly representing an F_1 assignment has to find a counterpart assembly representing an F_2 assignment to attach to. This attachment is done through the count state that represents the number of ones in the assignment. The affinities between separate count states are designed such that all assemblies can find a necessary counterpart if and only if $max\text{-}\mathbb{1}(F_1) = max\text{-}\mathbb{1}(F_2)$.

Theorem 5.2.5. *Unique Assembly Verification in One Dimensional freezing Affinity Strengthening Tile Automata is $P_{||}^{NP}$-complete.*

BIBLIOGRAPHY

[1] 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, 2002, pp. 23–32.

[2] P. K. AGARWAL, B. ARONOV, T. GEFT, AND D. HALPERIN, *On two-handed planar assembly partitioning with connectivity constraints*, in Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021, D. Marx, ed., SIAM, 2021, pp. 1740–1756.

[3] P. K. AGARWAL, B. ARONOV, T. GEFT, AND D. HALPERIN, *On two-handed planar assembly partitioning with connectivity constraints*, in Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2021, pp. 1740–1756.

[4] G. AGGARWAL, Q. CHENG, M. H. GOLDWASSER, M.-Y. KAO, P. M. DE ESPANES, AND R. T. SCHWELLER, *Complexities for generalized models of self-assembly*, SIAM Journal on Computing, 34 (2005), pp. 1493–1515.

[5] J. C. ALUMBAUGH, J. J. DAYMUDE, E. D. DEMAINE, M. J. PATITZ, AND A. W. RICHA, *Simulation of programmable matter systems using active tile-based self-assembly*, in DNA Computing and Molecular Programming, C. Thachuk and Y. Liu, eds., Cham, 2019, Springer International Publishing, pp. 140–158.

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

[7] N. BRYANS, E. CHINIFOROOSHAN, D. DOTY, L. KARI, AND S. SEKI, *The power of nondeterminism in self-assembly*, in Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2011, pp. 590–602.

[8] D. CABALLERO, T. GOMEZ, R. SCHWELLER, AND T. WYLIE, *Verification and Computation in Restricted Tile Automata*, in 26th International Conference on DNA Computing and Molecular Programming (DNA 26), C. Geary and M. J. Patitz, eds., vol. 174 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2020, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 10:1–10:18.

[9] D. CABALLERO, T. GOMEZ, R. SCHWELLER, AND T. WYLIE, *The complexity of multiple handed self-assembly*, in Unconventional Computation and Natural Computation, I. Kostitsyna and P. Orponen, eds., Cham, 2021, Springer International Publishing, pp. 1–18.

[10] 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.

[11] ——, *Complexity of verification in self-assembly with prebuilt assemblies*, in Proceedings of the Symposium on Algorithmic Foundations of Dynamic Networks, SAND'22, 2022.

[12] ——, *Unique assembly verification in two-handed self-assembly*, in International Colloquium on Automata, Languages and Programming (ICALP) 2022 (To Appear), 2022.

[13] S. CANNON, E. D. DEMAINE, M. L. DEMAINE, S. EISENSTAT, D. FURCY, M. J. PATITZ, R. SCHWELLER, S. M. SUMMERS, AND A. WINSLOW, *On the effects of hierarchical self-assembly for reducing program-size complexity*, Theoretical Computer Science, 894 (2021), pp. 50–78.

[14] S. CANNON, E. D. DEMAINE, M. L. DEMAINE, S. EISENSTAT, M. J. PATITZ, R. T. SCHWELLER, S. M. SUMMERS, AND A. 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), vol. 20 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, pp. 172–184.

[15] A. A. CANTU, A. LUCHSINGER, R. SCHWELLER, AND T. WYLIE, *Covert Computation in Self-Assembled Circuits*, in 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019), vol. 132 of Leibniz International Proceedings in Informatics (LIPIcs), 2019, pp. 31:1–31:14.

[16] ——, *Signal Passing Self-Assembly Simulates Tile Automata*, in 31st International Symposium on Algorithms and Computation (ISAAC 2020), Y. Cao, S.-W. Cheng, and M. Li, eds., vol. 181 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2020, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 53:1–53:17.

[17] ——, *Covert Computation in Self-Assembled Circuits*, Algorithmica, 83 (2021), pp. 531–552. arXiv:1908.06068.

[18] C. CHALK, D. FERNANDEZ, A. HUERTA, M. MALDONADO, R. SCHWELLER, AND L. SWEET, *Strict self-assembly of fractals using multiple hands*, Algorithmica, 76 (2014).

[19] C. CHALK, A. LUCHSINGER, E. MARTINEZ, R. SCHWELLER, A. WINSLOW, AND T. WYLIE, *Freezing simulates non-freezing tile automata*, in International Conference on DNA Computing and Molecular Programming, Springer, 2018, pp. 155–172.

[20] 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.

[21] G. CHATTERJEE, N. DALCHAU, R. A. MUSCAT, A. PHILLIPS, AND G. SEELIG, *A spatially localized architecture for fast and modular dna computing*, Nature nanotechnology, 12 (2017), pp. 920–927.

[22] H.-L. CHEN AND D. DOTY, *Parallelism and time in hierarchical self-assembly*, SIAM Journal on Computing, 46 (2017), pp. 661–709. Preliminary version appeared in SODA 2012.

[23] M. COOK, T. STÉRIN, AND D. WOODS, *Small tile sets that compute while solving mazes*, in 27th International Conference on DNA Computing and Molecular Programming, 2021.

[24] E. D. DEMAINE, M. L. DEMAINE, S. P. FEKETE, M. ISHAQUE, E. RAFALIN, R. T. SCHWELLER, AND D. L. SOUVAINE, *Staged self-assembly: nanomanufacture of arbitrary shapes with o(1) glues*, Natural Computing, 7 (2008), pp. 347–370.

[25] E. D. DEMAINE, M. L. DEMAINE, S. P. FEKETE, M. J. PATITZ, R. T. SCHWELLER, A. WINSLOW, AND D. WOODS, *One tile to rule them all: Simulating any tile assembly system with a single universal tile*, in International Colloquium on Automata, Languages, and Programming, Springer, 2014, pp. 368–379.

[26] D. DOTY, *Theory of algorithmic self-assembly*, Communications of the ACM, 55 (2012), pp. 78–88.

[27] ——, *Producibility in hierarchical self-assembly*, in Unconventional Computation and Natural Computation, O. H. Ibarra, L. Kari, and S. Kopecki, eds., Cham, 2014, Springer International Publishing, pp. 142–154.

[28] D. DOTY, L. KARI, AND B. MASSON, *Negative interactions in irreversible self-assembly*, Algorithmica, 66 (2013), pp. 153–172.

[29] N. FATES, *A guided tour of asynchronous cellular automata*, in International Workshop on Cellular Automata and Discrete Complex Systems, Springer, 2013, pp. 15–30.

[30] 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 Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2014, pp. 148–167.

[31] E. GOLES, P.-E. MEUNIER, I. RAPAPORT, AND G. THEYSSIER, *Communication complexity and intrinsic universality in cellular automata*, Theoretical Computer Science, 412 (2011), pp. 2–21.

[32] A. S.-A. R. GROUP, *VersaTile*. https://github.com/asarg/VersaTile, 2014.

[33] J. HENDRICKS AND J. OPSETH, *Self-assembly of 4-sided fractals in the two-handed tile assembly model*, Natural Computing, 18 (2018), pp. 75–92.

[34] J. HENDRICKS, M. J. PATITZ, T. A. ROGERS, AND S. M. SUMMERS, *The power of duples (in self-assembly): It's not so hip to be square*, Theoretical Computer Science, 743 (2018), pp. 148–166.

[35] A. KEENAN, R. SCHWELLER, M. SHERMAN, AND X. ZHONG, *Fast arithmetic in algorithmic self-assembly*, Natural Computing, 15 (2016), pp. 115–128.

[36] D. E. KNUTH AND A. RAGHUNATHAN, *The problem of compatible representatives*, SIAM Journal on Discrete Mathematics, 5 (1992), pp. 422–427.

[37] M. G. LAGOUDAKIS AND T. H. LABEAN, *2d dna self-assembly for satisfiability.*, in DNA Based Computers, 1999, pp. 141–154.

[38] P.-E. MEUNIER, , AND D. WOODS, *The non-cooperative tile assembly model is not intrinsically universal or capable of bounded turing machine simulation*, in STOC: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Association for Computing Machinery, 2017, pp. 328–341.

[39] P.-E. MEUNIER, M. J. PATITZ, S. M. SUMMERS, G. THEYSSIER, A. WINSLOW, AND D. WOODS, *Intrinsic universality in tile self-assembly requires cooperation*, in Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2014, pp. 752–771.

[40] P.-E. MEUNIER AND D. REGNAULT, *Directed Non-Cooperative Tile Assembly Is Decidable*, in 27th International Conference on DNA Computing and Molecular Programming (DNA 27), M. R. Lakin and P. Šulc, eds., vol. 205 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2021, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 6:1–6:21.

[41] P.-E. MEUNIER, D. REGNAULT, AND D. 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, Association for Computing Machinery, 2020, pp. 727–737.

[42] J. E. PADILLA, W. LIU, AND N. C. SEEMAN, *Hierarchical self assembly of patterns from the robinson tilings: Dna tile design in an enhanced tile assembly model*, Natural Computing, 11 (2012), pp. 323–338.

[43] J. E. PADILLA, M. J. PATITZ, R. T. SCHWELLER, N. C. SEEMAN, S. M. SUMMERS, AND X. ZHONG, *Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes*, International Journal of Foundations of Computer Science, 25 (2014), pp. 459–488.

[44] M. PATITZ, *Pytas*. http://self-assembly.net/wiki/index.php?title=PyTAS.

[45] M. J. PATITZ, *An introduction to tile-based self-assembly and a survey of recent results*, Natural Computing, 13 (2014), pp. 195–224.

[46] R. SCHWELLER, A. WINSLOW, AND T. WYLIE, *Complexities for high-temperature two-handed tile self-assembly*, in DNA Computing and Molecular Programming, R. Brijder and L. Qian, eds., Springer International Publishing, 2017, pp. 98–109.

[47] ——, *Verification in staged tile self-assembly*, Natural Computing, 18 (2019), pp. 107–117.

[48] N. C. SEEMAN, *Nucleic acid junctions and lattices*, Journal of theoretical biology, 99 (1982), pp. 237–247.

[49] H. SPAKOWSKI, *Completeness for parallel access to np and counting class separation*, Ausgezeichnete Informatikdissertationen 2005, (2006).

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

[51] E. WINFREE, R. SCHULMAN, AND C. EVANS, *The xgrow simulator*. https://www.dna.caltech.edu/Xgrow/.

[52] D. WOODS, *Intrinsic universality and the computational power of self-assembly*, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 373 (2015), p. 20140214.

[53] D. WOODS, D. DOTY, C. MYHRVOLD, J. HUI, F. ZHOU, P. YIN, AND E. WINFREE, *Diverse and robust molecular algorithms using reprogrammable dna self-assembly*, Nature, 567 (2019), pp. 366–372.

[54] T. WORSCH, *Towards intrinsically universal asynchronous ca*, Natural Computing, 12 (2013), pp. 539–550.

BIOGRAPHICAL SKETCH

Timothy Axel Gomez was born and grew up in the Rio Grande Valley and graduated from Harlingen High School South in 2016. Timothy received his Bachelor's in Computer Science from University of Texas Rio Grande Valley (UTRGV) in 2020 where he also served as president of Phi Kappa Theta, a fraternity on campus. While an undergraduate he joined the Algorithmic Self-Assembly research group where he along with other students published many papers working under Dr. Robert Schweller and Dr. Tim Wylie. Timothy continued his education at UTRGV where he received the Presidential Graduate Research Assistantship to support him as a graduate student. He received his Master's degree in Computer Science in 2022. Timothy can be reached at tagomez62@gmail.com