5-2018

Computational Complexity of Determining the Rigidity of FTAM Assemblies

Ian Perkins
University of Arkansas, Fayetteville

Computational Complexity of Determining the Rigidity of FTAM Assemblies

Matthew J. Patitz * Ian Perkins† Michael Sharp ‡

Abstract

In this paper, we discuss a tile-based self-assembly model called the Folding Tile Assembly Model (FTAM). We briefly define what makes the FTAM unique in its ability to have folding 2D tiles. We also discuss the difficulty of determining the computational complexity of certain FTAM properties despite it being simpler for less dynamic models. Specifically, we discuss the property of rigidity in FTAM assemblies by devising a simple definition of rigidity, so that it is easier to determine its complexity. We use a reduction between an assembly and a 3SAT instance along with a series of proofs to give a result that shows it is co-NP-complete to determine whether a given assembly is rigid.

1 Introduction

One of the biggest reasons for studying self-assembly and in particular folding in self-assembly is the importance it holds in the production of proteins. Despite there only being twenty essential amino acids in the human body, we can create tens of thousands of different proteins from those original twenty amino acids. A vast variety of these proteins can be attributed to the unique folding that occurs to a protein chain after translation. One goal for those interested in self assembly is to help shed light on these types of biological processes of protein folding by comparing them to the folding happening in the FTAM, an extension of Erik Winfree's abstract Tile Assembly Model (aTAM) [1]. Ideally, this would lead to insight as to how and why proteins fold in the way that they do.

2 Definitions

In this section we present definitions related to the Folding Tile Assembly Model (FTAM).

A tile type t in the FTAM is defined as a unit square that can be translated, rotated, and reflected throughout three dimensional space, but can only occupy a location such that its corners are positioned on four adjacent, coplanar points in $\mathbb{Z}^3$. Each tile type t has four sides $i \in \{N, E, S, W\}$, each with a "glue" that consists of a "label" $label_t(i)$, denoted as a string over some fixed alphabet Σ, a non-negative integer "strength" $str_t(i)$, and a boolean valued "flexibility" $flx_t(i)$. Only glues with matching strengths, flexibility, and complementary labels (as defined by the designer of the system) can bond together.

A *tile* is an instance of a tile type. In addition to the type, it has coordinates, an orientation, an inversion, and a rotation. The coordinates are just a triple of integers that corresponds to a point in $\mathbb{Z}^3$. The orientation is just an x, y, or z that corresponds to the normal of the tile. The inversion is a true or false value that corresponds to whether the normal is pointed in the positive or negative direction. The rotation is a value between 0 and 3 that corresponds to the number of 90 degree clockwise turns the tile took to attach to the assembly. An important note is that we only allow tiles with the same inversion to bind together, unless

*Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR, USA patitz@uark.edu This author's research was supported in part by National Science Foundation Grant CCF-1422152 and CAREER-1553166.

†Department of Computer Science and Computer Engineering, University of Arkansas irperkin@uark.edu

‡Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR, USA. mrs018@uark.edu. This author's research was supported in part by National Science Foundation Grants CCF-1422152 and CAREER-1553166.

they are folding through a bent flexible bond, in which case they must have the same inversion if the flexible bond was straight.

We define an *assembly* as a graph whose vertices are tiles and whose edges represent bound glues between adjacent edges of two tiles. Each tile has four ports (N,E,S,W) from which edges can extend, and since each edge represents a pair of complementary bound glues, it is either rigid or flexible, based on the glue type.

We define a *face* to be a set of coplanar tiles that are all bound together through rigid bonds. We define a *face graph* to be an inflation of the assembly graph where every maximal subgraph in which every node can be reached from every other node using a path of rigid tiles is replaced by a single node in the face graph. Two nodes in the face graph that correspond to two groups of nodes in the assembly graph have an edge if and only if there is at least one flexible bond between the any single node in the first group of the assembly graph and any single node in the second group of the assembly graph.

Given an assembly α, a *configuration* χ is a mapping from every flexible bond in α to an orientation from "Up", "Down", "Straight". An orientation for a flexible bond describes which way a flexible tile is folding in relation to the other tiles it is attached to. Specifically, we define a straight bond as a bond between two tiles where the normal vectors are pointing in the same direction. An up bond refers to the bond between two tiles where the normal vectors for the tiles intersect, and a down bond refers to a bond between two tiles where the inverse of their normal vectors intersect.

An *embedding* ϵ of α is a mapping from each tile to a placement. Given an assembly and a configuration, we can obtain an embedding by choosing an initial tile and assigning it a placement and computing the placement of each additional tile according to how it is bonded with tiles that are already placed. We say a configuration χ_α is *valid* if and only if the embedding obtained from the configuration (1) does not place more than one tile at any tile location, (2) does not have more than one bond per tile edge, and (3) does not have contradicting bond loops. To clarify, contradicting bond loops occur when placing a loop of tiles that are all bound in a loop causes the last tile to be placed at a location that is not adjacent to the first tile, therefore making the loop unable to close. Note that two embeddings that use different initial tiles and initial placements but the same configuration will be equivalent up to rotation and translation.

3 Rigidity from Assembly

We define a rigid assembly as an assembly with only one configuration or two configurations where the two configurations are chiral versions of each other. Alternatively, if there is another configuration for the assembly other than the chiral configuration, the assembly is said to be flexible.

Configurations of assemblies in the FTAM are also subject to chirality. A configuration of an assembly can be converted to its chiral version by reversing all of its up bonds to down bonds and vice versa while leaving the straight bonds as they are. Since this is another configuration the assembly can be in that differs from the initial configuration, this would make every assembly flexible. For this reason, we do not consider the chiral configuration when determining rigidity for the assembly and configuration as it is a trivial case.

3.1 Rigidity from Assembly is in co-NP

Theorem 3.1. Given an assembly α, determining if α is rigid is in co-NP

Take an instance of an assembly α and a certificate χ. Our certificate in this case will be two distinct configurations, c and c', for assembly α. To determine whether our certificate is valid we must check three aspects of the certificate: (1) c and c' are valid encodings of configurations, (2) ensure that c and c' are distinct, and finally (3) project the assembly α to be in each configuration, c and c', to make sure they are *valid* configurations.

To determine if the configurations are encoded properly, both configurations must map every flexible bond in α to an orientation (up, down, or straight). An orientation for a flexible bond describes which way a flexible tile is folding in relation to the other tiles it is attached to. Specifically, we define a straight bond as a bond between two tiles where the normal vectors are pointing in the same direction. An up bond refers to the bond between two tiles where the normal vectors for the tiles intersect, and a down bond refers to a bond

Figure 1: The three different orientations a flexible bond can be in: straight, up, or down.

between two tiles where the inverse of their normal vectors intersect. For example, in figure 1 we can see the three different orientations a flexible bond can have. We can check for valid encodings of configurations in linear time with respect to how many flexible bonds there are in α. Similarly, to determine if c and c' are distinct, we can compare each flexible bond from c and c' in linear time with respect to the number of flexible bonds.

As for determining whether c and c' are valid configurations after projecting it onto α, we must first define what a *valid* configuration is. A valid configuration is one in which (1) no two tiles in α occupy the same space, (2) doesn't bond tiles through the same space, and (3) does not have a contradicting bond loop. To elaborate, a contradicting bond loop occurs when the last tile in a loop of tiles is not placed adjacent to the first tile in the loop, thereby leaving the loop unable to close. Checking these conditions requires takes linear time with respect to the number of tiles in α.

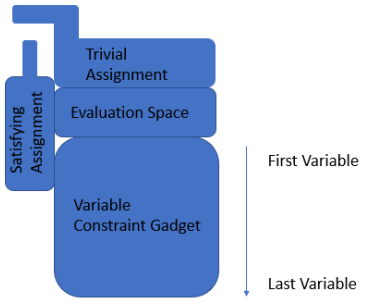
If all three conditions are satisfied, then c and c' are both valid configurations for α and the assembly is said to be flexible. Therefore, flexibility is said to be in NP as all checks to determine if the certificate is valid can be done in linear time, and the size of the certificate is polynomial in size since the certificate is only as large as the number of tiles in the assembly α. Note that from our prior definition of flexibility and rigidity, an assembly cannot both be rigid and flexible. Either there is an alternative, valid configuration for the assembly and the assembly is said to be flexible or there is no alternative, valid configuration and the assembly is said to be rigid. As such, this means flexibility and rigidity are complementary languages of each other. Therefore, rigidity is said to be in co-NP since its complement, flexibility, is in NP.

3.2 The Complement of Rigidity from Assembly is NP-hard

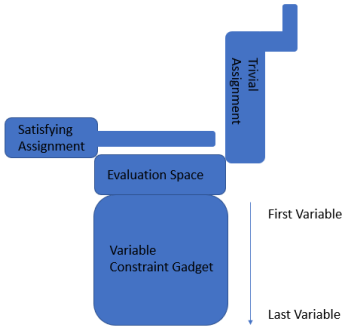
To determine the computational complexity of determining rigidity from an FTAM assembly, we will reduce 3SAT to the complement of the rigidity from assembly problem. We do this by encoding a 3SAT formula as an FTAM assembly with some configuration such that a valid, non-chiral configuration exists if and only if the 3SAT formula is satisfiable. For this reduction, we have developed the *3SAT Solver*. The 3SAT solver is composed of four main pieces: the evaluation space, the satisfying assignment hat, the trivial assignment hat, and the variable constraint gadgets. A high level diagram of all four parts working together can be seen in figure 2.

Before we delve into more detailed descriptions of each of the main parts of the 3SAT solver, we define a few tools for use later in the overall proof. We define a set of faces as a set of coplanar tiles that are all bound together through rigid bonds. We define *entanglement* as the relationship of certain connections in a set of faces to invert and cause all other connections to invert with it. We define a *traditional 4-sided loop* as a cycle of faces where each face is attached to two other faces on its opposite ends. Finally, we define a *rigid component* as a set of faces that have been entangled (note: a traditional 4-sided loop will always be a rigid component).

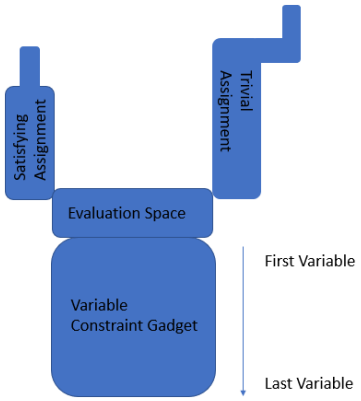
Claim 3.1. Given an assembly α and a configuration χ, for any rigid component c_{rigid} in α, if α has another valid, non-chiral configuration χ' that it can exist in, then there must also exist another valid configuration χ'' in which the rigid component c_{rigid} is not inverted.



(a) The initial trivial state that every assembly can be in. The trivial state ensures that the assembly is rigid if there is no solution to the inputted 3SAT problem.



(b) The satisfying state whereby the assembly indicates that it has found a solution to the input 3SAT problem by pressing the satisfying assignment hat against the evaluation space.



(c) An invalid configuration in which neither the satisfying nor trivial assignment hat is down.

Figure 2: The two different states the 3SAT solver can be in and an invalid state

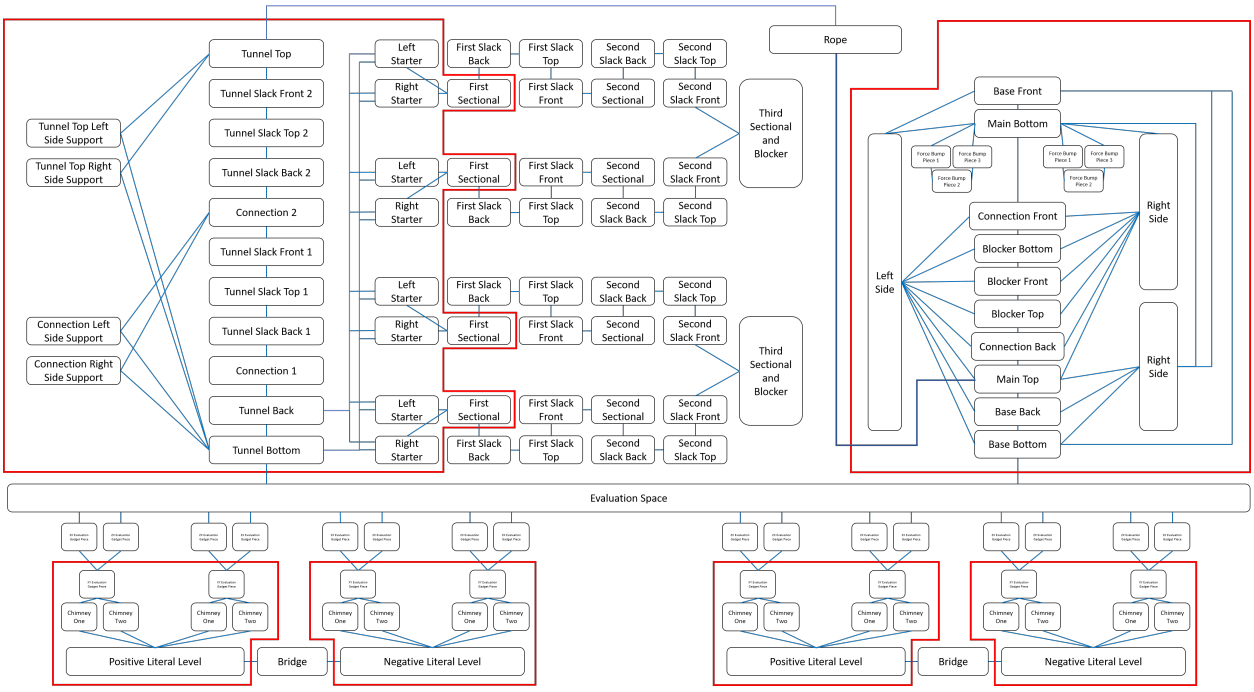


Figure 3: A face graph of the 3SAT solver and its associated connections. Each group of nodes surrounded by red indicates a rigid component.[1]

Proof. Assuming c_{rigid} is not inverted in χ', then $\chi' = \chi''$. However, if we assume c_{rigid} is inverted, then χ'' is simply the inversion of χ'. □

Claim 3.2. If two rigid components, c_1 and c_2 share two faces p_1 and p_2 that are not coplanar, then c_1 and c_2 can be entangled into one rigid component.

Proof. Assume that c_1 can actually reorient relative to c_2 without having c_2 reorient as well. Since both c_1 and c_2 are defined as rigid, the only way either can reorient is by inversion. If the inversion occurs over p_1, then p_2 will have reoriented over the plane that p_1 exists in. Since p_2 is part of c_2, the only reorientation it could have made was an inversion. If an inversion occurs over another face, then the plane the inversion occurs on can only contain either p_1 or p_2 but not both since by definition they are not coplanar. Since the inversion occurred on only p_1 or p_2, that means one of the planes still needed to reorient. Both cases of reorientation lead to contradictions, so c_1 and c_2 must be one, single rigid component. □

With these tools in mind, we will discuss the four main parts of the 3SAT solver and their accompanying proofs. In the following sections, we will describe each of the four main parts of the 3SAT solver, their purpose, and determine with the help of some proofs and figure 3 that a majority of these components can be described as a single rigid component formed through entanglement and the presence of traditional 4-sided loops. Lastly, we will discuss how the remaining flexible tiles of the assembly interact with each other in order to solve 3SAT problems.

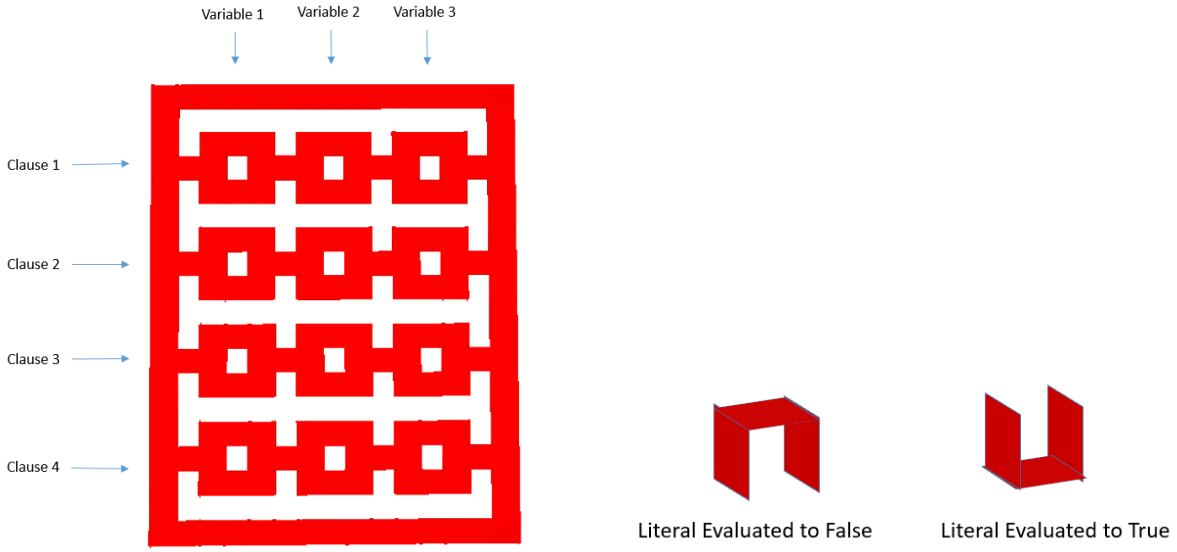
3.2.1 Evaluation Space

The evaluation space, shown in figure 4a, is a frame that facilitates interactions between the variable constraint gadgets, trivial assignment hat, and satisfying assignment hat. The frame consists of any number of rows each containing three flat squares with an empty center connected by a line of tiles. Each row represents a clause in a 3SAT problem while each empty center of the flat square represents a variable in said clause.

[1]Credit for this figure goes to Michael Sharp

Based on the input to the 3SAT instance, evaluation gadgets will form on the empty center of these squares. These evaluation gadgets consist of only three tiles total. Of these three tiles, the two tiles that occupy the ZX plane have flexible bonds while the third tile in the XY plane is rigid. This is what allows the evaluation gadgets to only have two orientations, the true orientation or the false orientation as seen in figure 4b. The evaluation gadgets appear on the evaluation space when there are at least two instances of a particular variable where at least one instance of the variable is negated ($\overline{x}$) and at least one other is not negated (x). An evaluation gadget that is popped up indicates the literal evaluated to "False," and an evaluation gadget popped down indicates the literal evaluated to "True." It should be noted that these evaluation gadgets only appear on the evaluation space when there at least two instances of a variable where at least one variable is negated and the other is not. In other words, for variables with instances that are either all negated or all not negated, an evaluation gadget will not form.

The evaluation space also serves to connect the trivial and satisfying assignment hats. Both the trivial and satisfying assignment hats are attached to each long end of the evaluation space by a line of flexible tiles. This allows for either hat to come down and press against the evaluation space depending on whether there is a solution to the inputted 3SAT formula. In the case of a solution to the 3SAT formula, the satisfying assignment hat will come down like in figure 2b; otherwise, the trivial assignment hat stays down and ensures that the assembly is rigid like in figure 2a.



(a) The evaluation space with three variables per clause and, in this case, four total clauses.

(b) Two different orientations the evaluation gadgets can be in.

Figure 4: (a) The evaluation space and (b) two evaluation gadgets.

3.2.2 Satisfying Assignment Hat

Of the two assignment hats that are part of the assembly, the satisfying assignment hat serves to determine whether there is a solution to the input 3SAT problem. The satisfying assignment will come down and press against the evaluation space if and only if there is a solution to the input 3SAT problem. The satisfying assignment hat accomplishes this by using a *checker* that extends across each clause in the evaluation space. There is one checker per clause and the checker can only extend to three different positions for each of the different variables in each clause, as seen in figure 5a. The checkers can only appear in three different positions due to the enclosed tiles that house each satisfying assignment checker and the eight flexible bonds (labeled in orange in figure 5c) that compose the checker. In figures 5b and 5c, we see a side view of the

checker surrounded by a rigid structure of tiles. The structure that surrounds the checker has two large tunnels that serve to contain a part of the folded checker whenever the checker needs to shorten its length. To reach the farthest variable, the checker will be totally extended. For the center variable, part of the checker will be folded into one of the tunnels, and for the closest variable, the checker will be folded into both of the tunnels. No other length is possible due to the construction of the tiles that encase the checker. If the checker were to attempt a different configuration than the ones listed, the checker would collide with its encased structure. When the satisfying assignment hat is down, the checkers will extend across their corresponding clauses to the variable instance that is part of the solution to the input 3SAT problem which can be seen in figure 9c.

Claim 3.3. The satisfying assignment hat cannot reorient without being inverted.

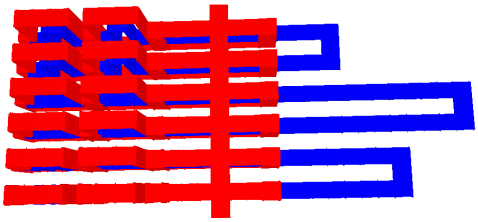
Proof. Using figure 3 and 5c as a reference, notice that the Tunnel Bottom, Tunnel Top Right Side Support, Tunnel Top, and Tunnel Top Left Side Support can be used to form a traditional 4-sided loop. The Tunnel Bottom, Connection Right Side Support, Connection 2, and Connection Left Side Support also forms a traditional 4-sided loop. If only one of them were to reorient and invert, then the inverting rigid component would collide with the non-inverting rigid component. For this reason, we can say that these two rigid components are entangled. Now, since the two rigid components we described are actually entangled as one larger rigid component, we can also say that the tunnel slack 2 pieces are also entangled since they connect to the Connection 2 and Tunnel Top pieces.

Now, note that using either the Left Starter or Right Starter with the Tunnel Bottom and Tunnel Back forms a vertex. Due to the vertex, Tunnel Back can only be oriented as up or down but never straight. If Tunnel Back does invert to the down position, then it would be 4 units away from its original orientation. In addition, the Tunnel Back would also be another 4 units away from the Connection 2 and Tunnel Slack Front 1 pieces leaving us with a total Manhattan distance of 8. Since the length of Connection 1, Tunnel Slack Back 1, Tunnel Slack Top 1, and Tunnel Slack Front 1 total to 8 as well, that means for the connection from these pieces to the Tunnel Back will require using its entire length to connect. However, this would cause the tiles to collide with either the Tunnel Bottom or Tunnel Back, so it is not possible. For this reason, we can say that Tunnel Back, Left Starters, Right Starters, and First Sectionals are all entangled into one rigid component. This reduces the Manhattan distance needed to be covered to 4. Tunnel Front and Tunnel Back are made of two tiles, Tunnel Top is composed of three tiles, and the connection between the two tunnels is only one tile. This leaves us with an arrangement where the Tunnel Front and Back's length cancels each other out while the length of the Tunnel Top and the Connection 1 total to 4 units to make up the distance needed. Therefore, this leaves us with two options. Either the back and front slack folded towards the Tunnel Bottom in which case this would collide with the Tunnel Slack Top 1 or the slack folds away from the Tunnel Bottom and we are simply left with one rigid component. □

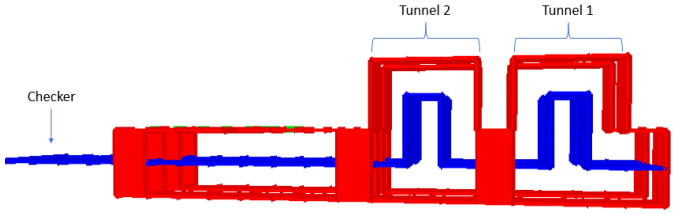
3.2.3 Trivial Assignment Hat

The trivial assignment hat, seen in figure 6a and 6b is one of the two hats that can rotate downward and press down against the evaluation space. The trivial assignment hat serves to keep the entire assembly rigid if and only if there is no solution to the inputted 3SAT problem. The trivial assignment hat forces the evaluation gadgets on the evaluation space by using the force bumps that form on the hat to press down on the evaluation gadgets and force them into a false orientation. These force bumps, which are different in purpose than the evaluation gadgets, form for every corresponding negative literal on the evaluation space. This effectively assigns the value false to all variables in the formula.

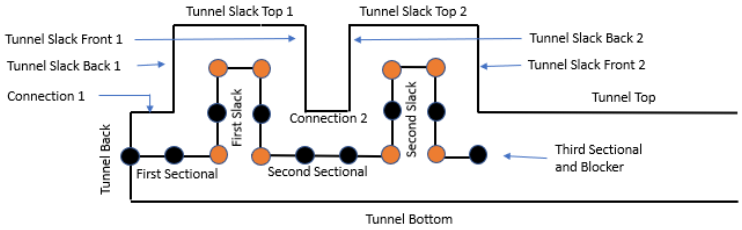
In addition to keeping the evaluation gadgets and evaluation space rigid, the trivial assignment also serves to keep the satisfying assignment hat in place. At the top of the trivial assignment hat is a bracket-like protrusion that serves to block the satisfying assignment hat. When the trivial assignment hat is pressed against the evaluation space, the bracket attached to the trivial assignment hat reaches past the evaluation space and serves to block the satisfying assignment checkers. This prevents the checkers from extending and providing an alternative configuration that does not solve the inputted 3SAT instance.



(a) The three different positions the checker can be in



(b) Side view of the tunnels the checker can reside in when it needs to shorten its length



(c) Labeled satisfying assignment hat showing the names of each of the faces that correspond to the face graph in figure 3. Each of the orange circles indicate a flexible bond.

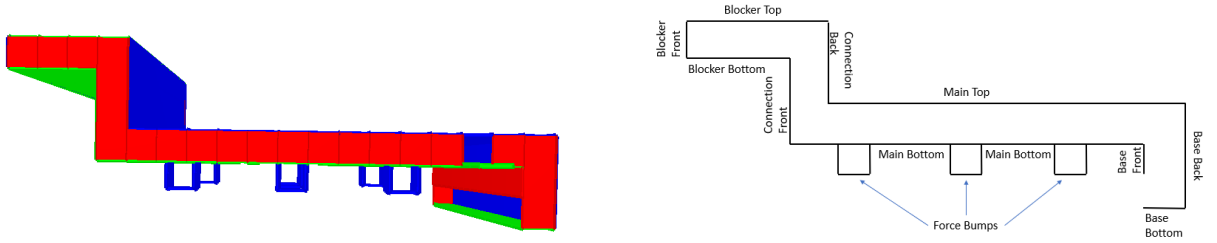
Figure 5

Claim 3.4. The trivial assignment hat cannot reorient without being inverted.

Proof. Using figure 3 as reference, we show that most faces in the trivial assignment hat are part of a traditional 4-sided loop. For the most part, these loops are created with the Left Side and one of the two Right Side pieces that compose the trivial assignment hat. Specifically, the loops will connect Left Side to part one, part one to Right Side, Right Side to part two, and part two back to Left Side. The actual pieces being replaced by parts one and two respectively include: (Base Bottom, Main Top), (Base Front, Base Back), (Main Bottom, Main Top), (Main Bottom, Blocker Top), (Connection Front, Connection Back), (Blocker Front, Blocker Back), (Blocker Bottom, Blocker Top). Using claim 3.2, we can see that we are left with two rigid components, one for each of the right sides. However, since each of these rigid components are also connected to the Left Side, Main Bottom, and Main Top, we can say that these two rigid components are entangled and therefore are actually one large rigid component. As for the force bumps that form on the trivial assignment hat, they each form a traditional 4-sided loop with the Main Bottom that they are attached to. If these force bumps were to ever reorient and invert, they would be blocked by the Main Top of the trivial assignment hat. Therefore, each force bump is also entangled with the rest of the trivial assignment hat making it one large rigid trivial assignment hat. □

3.2.4 Variable Constraint Gadget

The variable constraint gadget seen in figure 7 serves to keep all variables in agreement. We accomplish this by having one variable constraint gadget per "mixed" variable, a variable with at least one non-negated instance and at least one negated instance (x and $\overline{x}$). Each variable constraint gadget occupies a certain Z level below the evaluation space that corresponds to a particular "mixed" variable in the 3SAT problem. The variable constraint gadget is composed of the bonds between the evaluations space, the evaluation gadgets, variable constraint levels, and bridges. A parallel line of tiles called *chimneys* connect to the evaluation gadgets corresponding to that positive/negative variable and extend downwards to the appropriate variable constraint level for that positive/negative variable. For example, all evaluation gadgets that correspond to x literals have chimneys that extend to the variable constraint level for x and all evaluation gadgets that correspond to $\overline{x}$ have chimneys that extend to the the variable constraint level that corresponds to $\overline{x}$. Then,



(a) The trivial assignment hat in the FTAM visualizer.

(b) Labeled trivial assignment hat showing the names of each faces that correspond to the face graph in figure 3

Figure 6

these line of tiles that span the XY plane converge at a bridge as seen in figure 7. The bridge consists of two tiles in the ZX plane that serves to guarantee that the plane the positive instances of variables are connecting to is two units away in the Z direction to the plane that the negative instances of the same variable are connecting.

Claim 3.5. The variable constraint gadgets cannot reorient unless truth values are changed or it inverts.

Proof. For reference, please direct yourself to figure 3 for this proof. Recall that a variable constraint gadget will only appear when an instance of some variable, say x, has at least one instance of a positive literal and at least one instance of a negative literal. For every positive instance of x there will be a line of tiles (i.e. a chimney) attached to the respective evaluation gadget extending down to the appropriate variable constraint level for x. The same goes for all negated instances of x except the line of tiles will extend to the appropriate variable constraint level for $\overline{x}$ instead.

First, notice that starting from the XY tile to one side of the chimney to either the positive or negative variable constraint levels and back up the other side of the chimney to the initial XY piece creates a traditional 4-sided loop. Since this occurs for every evaluation gadget on the evaluation space, we know that from the XY tile down to the positive and negative levels of the variable constraint gadget are rigid with the exception of the bridge that connects each of the levels.

Now, we have two entangled rigid components per variable constraint gadget. The only piece that can cause a reorientation other than an inversion is the two ZX tiles attached to the initial XY tile. Note that for the ZX tiles to connect to the XY tile both ZX tiles must be in the same orientation (up, up or down, down), so that they can have a gap in between the two tiles for the XY tile to attach. With that in mind, these two ZX tiles can cause a reorientation that moves the rigid pieces below it either up two units or down two units and moves the other complementary rigid piece in the other direction by two units. This is the only reorientation that can happen because, as we have already determined, from one ZX tile to the next the variable assignment gadget is rigid. If a ZX tile were to reorient, it would cause all other ZX tiles connected to its respective level (x or $\overline{x}$) to reorient as well. Also, notice that both the x level and the $\overline{x}$ level cannot be in the same plane as they would be incident with each other and have no room for the bridge to connect. Therefore one of the rigid pieces must have its ZX tiles from the evaluation gadgets in the up orientation and the other rigid piece will need its ZX tiles to be in the down orientation for both levels to connect to the bridge. Since all variable constraint gadgets are the same, this applies to all variable constraint gadgets. □

3.2.5 3SAT Solver

Finally, we introduce the last piece, the *rope*. The rope is what attaches the trivial assignment hat and satisfying assignment hat to avoid an invalid configuration like in figure 2c. The rope consists of a 15-by-1 line of tiles that attaches from the Main Top of the trivial assignment hat to the Tunnel Top of the satisfying assignment hat. This rope's normal points in the positive Z direction while the normal of the evaluation space is pointed in the negative Z direction. Note that this rope creates a loop starting with the trivial

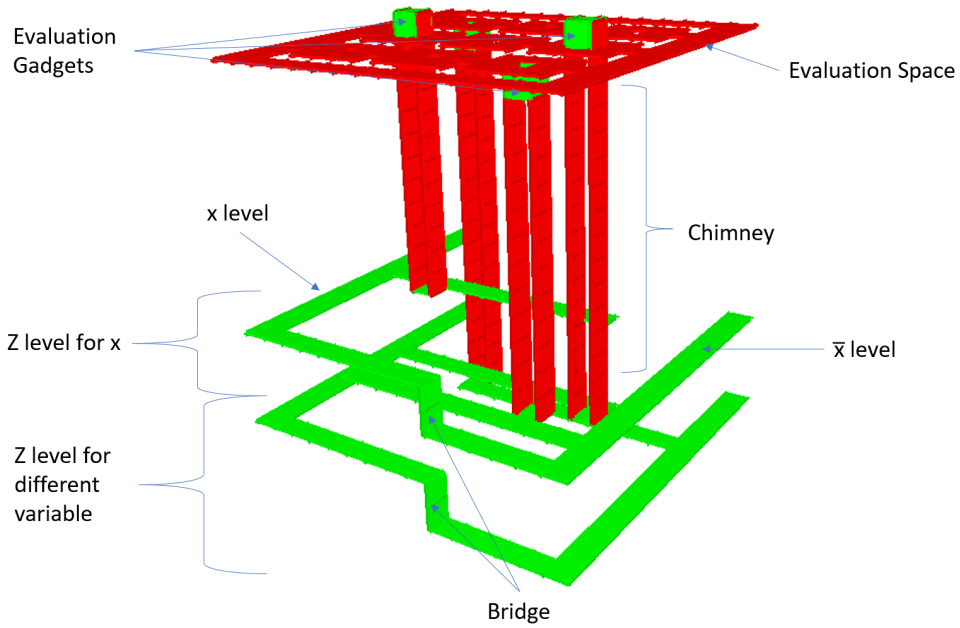


Figure 7: Variable constraint gadget connected to the evaluation space by the evaluation gadgets.

assignment hat attached to the evaluation space, the evaluation space attached to the satisfying assignment hat, the satisfying assignment hat to the rope, and the rope back to the trivial assignment hat. We will refer to this loop from here on out as the *main loop* of the 3SAT solver.

With this, we can focus on the flexible bonds in the assembly and describe how they move relative to one another. The flexible bonds left include the main loop previously discussed, the checkers that are part of the satisfying assignment hat, and the ZX tiles connected to the variable constraint gadgets.

Each checker in the satisfying assignment hat is composed of two eight tile lines that are connected by a final piece known as the Third Sectional and Blocker piece. Both of the eight tile lines that compose one checker must be in the same orientation of bonds so that they can connect to the Third Sectional and Blocker piece. This requires the two eight tile lines to be at the same X coordinate, which means their orientations must be identical. As mentioned in section 3.2.2, the checker can only have three orientations due to the structure it is encased in. Besides the designed three positions the checker can be in, all other positions would cause the checker to collide with its surrounding structure.

The variable constraint group as described in section 3.2.4 is composed of the bonds between the evaluation space, the evaluation gadgets, chimneys, variable constraint levels, and bridges. As we previously discussed in claim 3.5, the variable constraint gadget can have only two states, one where the variable is false and one where the variable is true. This means that the number of possible configurations can only be as many as there are possible assignments for each variable, or 2^n where n is the number of variables in the 3SAT formula.

Lastly, we take a look at the main loop when it is in the trivial state or the satisfying state.

Claim 3.6. Neither the trivial nor the satisfying assignment hat can invert without the other.

Proof. In figure 8 we see the bonds between the rope, satisfying assignment hat, trivial assignment hat, and evaluation space in two possible configurations, the trivial state and the satisfying state. Note that the displacement vectors between either assignment hat and where they attach to the evaluation space as well as the bond where the same hat connects to the rope is some rotation of the vector $(2, 3)$ in the ZX plane. If one of the hats were to reorient without the other, the displacement vector of the inverted hat would be some rotation of the vector $(3, 2)$. If this inversion were to happen, the rope that connects the satisfying assignment hat to the trivial assignment hat would be off by one unit in either the positive or negative Z direction. These displacement vectors for each of the assignment hats must be in the same Z plane so that

they can be connected by the rope. For this to happen, the X values must be a negation of each other. This results in four possible distances (9, 11, 19, or 21) for the rope to cover in order for one of the hats to invert without the other. These four possible distances are determined by taking the length of the evaluation space, 15, and adding or subtracting both of the possible X coordinates of the displacement vectors, 2 or 3. Since none of the possible distances equal 15, the length of the rope, neither of the assignment hats can reorient without the other.

□

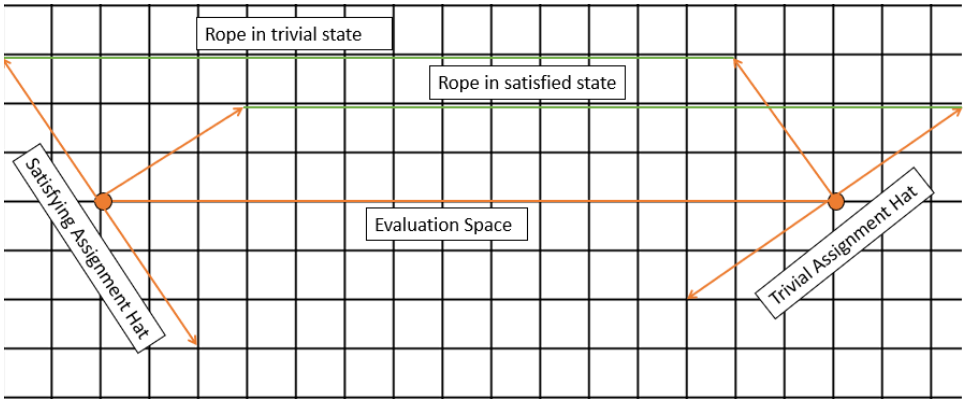


Figure 8: The main loop viewed from the negative Y direction. This figure illustrates that there can only be two positions the assignment hats can be in such that they're still connected to the rope and to the evaluation space.

Claim 3.7. If the main loop is in the trivial state, no other bond can reconfigure.

Proof. Firstly, we can easily see in figure 9 that the variable constraint gadgets cannot invert without the satisfying assignment hat and trivial assignment hat. If the variable constraint gadgets were to invert while the trivial assignment hat was pressing down on the evaluation space, the chimneys that connect the evaluation gadgets to the appropriate variable constraint levels would collide with the Main Bottom and Main Top pieces of the trivial assignment hat. Therefore, the variable constraint gadget will not reorient unless the other main components are reorienting as well. The bonds in the variable constraint gadget are also forced into a particular configuration when the trivial assignment is pressed down against the evaluation space. When the trivial assignment hat is down, the bumps on the trivial assignment hat forces all evaluation gadgets for positive literals to be popped down. This causes the entire variable constraint gadget to reorient. The "popped" evaluation gadget causes its respective variable constraint levels down, thereby forcing the variable constraint level for negative literals to reorient and pop its evaluation gadgets up.

If you recall in section 3.2.3, we discussed that when the trivial assignment hat is down, a bracket that is part of the trivial assignment hat extends past the evaluation space to block the checkers that would otherwise be extending in the positive Z direction. This limits the checkers from reorienting to anything other than its shortest length where both tunnels are filled with slack from the checker.

Therefore, if the main loop bonds cannot reorient from the trivial state, the entire assembly is rigid. □

Claim 3.8. If the main loop is in the satisfied state, the other free flexible bonds can only configure in a way such that no tiles in the assembly are overlapping if and only if there is a satisfying assignment to the corresponding 3SAT formula.

Proof. For this proof, we can ignore the inversions of the variable constraint gadgets when the main loop is in the satisfied state. If the variable constraint gadgets were to invert using the XY tile on the evaluation gadgets, this would not affect the functionality of the variable constraint gadget. The inversion of the XY tile

would simply cause all positive literals of one variable to assume one truth value while all negative literals would assume the other. For this reason, we ignore the potential inversions of variable constraint gadgets.

Starting with the forward portion of the biconditional, we have "If the assembly configures such that no tiles overlap, there is a satisfying assignment." In the satisfied state, the Third Sectional and Blocker of each checker occupies a space one unit in the positive Z direction of one of the three squares on the evaluation space that corresponds to a certain variable in the input 3SAT formula. If no tiles overlap, then one of the literals in the clause has evaluated to true, so it is popped down allowing the checker to occupy the space above it. To start building the satisfying assignment, we determine whether the literal that evaluated to true represented a positive literal or a negative literal. If the positive literal evaluated to true, then we set that variable to be true as part of the satisfying assignment. Alternatively, if it was a negative literal that evaluated to true, then we set that variable to false. As for the variables that did not have an assignment set to true or false, we can assign either truth value. We continue this process for each of the checkers until we have a complete variable assignment that satisfies every clause in the 3SAT formula.

The reverse of the biconditional states, "If there is a satisfying assignment, the assembly can configure such that no tiles overlap." When we have a satisfying assignment, we can take every variable with a true assignment, raise the variable constraint level, and pop the corresponding evaluation gadget in the positive Z direction that corresponds to the negative literals of that variable. Then we can lower the variable constraint level and pop the evaluation gadgets down in the negative Z direction that corresponds to the positive literals of that variable. On the other hand, if the variable is assigned a false value, we do the opposite. Since we know we have a satisfying assignment, one evaluation gadget in each clause must be popped down allowing the checker to occupy the space above the evaluation gadget that does not overlap with any other tile.

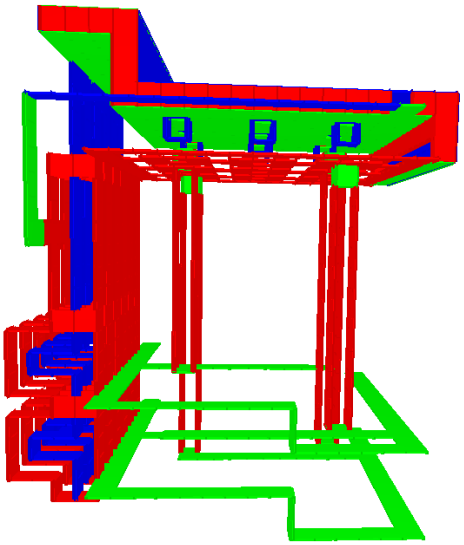
□

After taking a look at each of the four main parts as well as the rope, we can finally put them together to create the 3SAT solver. Both the satisfying assignment hat as well as the trivial assignment hat are attached to the evaluation space by a line of flexible tiles. The variable constraint gadgets are beneath the evaluation space connected to the ZX tiles from the evaluation gadgets by their chimneys. Finally, the rope that was previously discussed attaches the trivial assignment hat to the satisfying assignment hat to ensure the assembly cannot take an invalid configuration like in figure 2c. This creates the 3SAT solver which can be seen in figure 9.

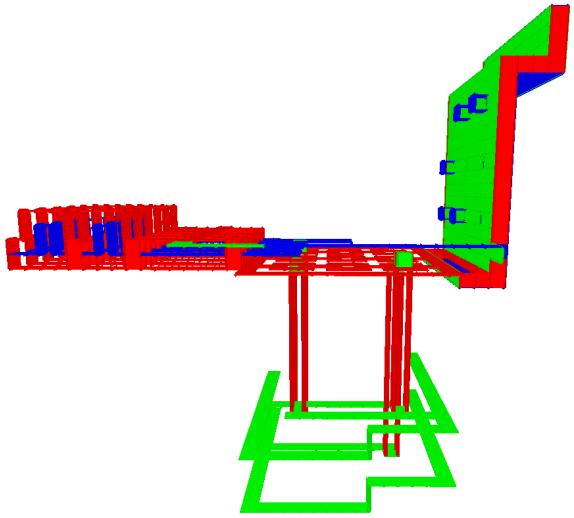
Altogether, we know by claim 3.8, if the 3SAT formula has a satisfying assignment, then the machine has at least one additional configuration in the satisfied state it can configure to. Otherwise, if the corresponding 3SAT formula does not have a satisfying assignment, there is no additional configuration in the satisfied state. By claim 3.7, there is only one configuration when in the trivial state. Therefore, determining if the assembly has at least one or more valid configurations, the complement of rigidity from assembly, is polynomial time reducible from 3SAT and therefore NP-hard.

4 Conclusion

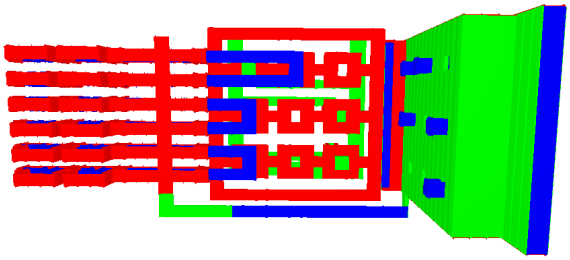
In this paper, we introduced the Flexible Tile Assembly Model (FTAM). We discussed rigidity's complementary properties to flexibility in section 3.1 and determined rigidity was in co-NP. In section 3.2, we discussed the high level overview of the designed 3SAT solver and how it is intended to show that determining the complement of rigidity from an assembly is NP-hard. In subsections 3.2.1 to 3.2.4, we discussed each of the four main parts that made up the 3SAT solver and discussed why each of the main pieces can be considered one large rigid piece relative to the other flexible pieces in the assembly. Afterwards, in section 3.2.5 we discussed how the four main pieces come together to form an assembly that solves 3SAT instances proving that determining the complement of rigidity from an assembly is NP-hard. Therefore, from sections 3.1 and 3.2, we have deduced that given an assembly it is co-NP-complete to determine whether an assembly is rigid.



(a) A side view of the 3SAT solver when the trivial assignment hat is down.



(b) A side view of the 3SAT solver when the satisfying assignment hat is down.



(c) A top view of the 3SAT solver when the satisfying assignment hat is down indicating there is a solution to the input 3SAT problem.

Figure 9: The 3SAT Solver

References

[1] Erik Winfree, Algorithmic Self-Assembly of DNA, Ph.D. thesis, California Institute of Technology, June 1998.