Self-assembly is a process which is ubiquitous in natural, especially biological systems. It occurs when groups of relatively simple components spontaneously combine to form more complex structures. While such systems have inspired a large amount of research into designing theoretical models of self-assembling systems, and even laboratory-based implementations of them, these artificial models and systems often tend to be lacking in one of the powerful features of natural systems (e.g. the assembly and folding of proteins), which is dynamic reconfigurability of structures. In this paper, we present a new mathematical model of self-assembly, based on the abstract Tile Assembly Model (aTAM), called the Flexible Tile Assembly Model (FTAM). In the FTAM, the individual components are 2-dimensional tiles as in the aTAM, but in the FTAM, bonds between the edges of tiles can be flexible, allowing bonds to flex and entire structures to reconfigure, thus allowing 2-dimensional components to form 3-dimensional structures. We analyze the powers and limitations of FTAM systems by (1) demonstrating how flexibility can be controlled to carefully build desired structures, and (2) showing how flexibility can be beneficially harnessed to form structures which can “efficiently” reconfigure into many different configurations and/or greatly varying configurations. We also show that with such power comes a heavy burden in terms of computational complexity of simulation and prediction by proving that for important properties of FTAM systems, determining their existence is intractable, even for properties which are easily computed for systems in less dynamic models.

Keywords  
Self-assembly · 3-Dimensional · Folding · Reconfiguration

1 Introduction

Proteins are a fantastically diverse set of biomolecules, with structures and functions that can vary wildly from each other, such as fibrous proteins (like collagen), enzymatic proteins (like catalase), and transport proteins (like hemoglobin). Truly amazing is the fact that such diversity arises solely from the linear combination of a series of copies of only 20 amino acid building blocks. It is the specific sequence of amino acids, interacting with each other as they are combined, which causes each chain to fold in a specific way and each protein to assume its particular three-dimensional structure, and this in turn dictates its structural and functional properties. Inspired by the prowess of nature to build molecules with such precision and heterogeneity, scientists have studied the mechanisms of protein folding—to realize that the dynamics are so complex that predicting a protein’s shape given its amino acid sequence is considered to be intractable (Fraenkel 1993; Crescenzi et al. 1998), and engineers have begun to develop artificial systems which fold self-assembling...
molecules into complex structures (Rothemund et al. 2005; Rothemund 2006; Barish et al. 2009; Liu et al. 2011)—but with results that to date still lack the diversity of biology.

In order to help progress understanding of the dynamics of systems which self-assemble out of folding components, and to provide a framework for studying such systems, in this paper we introduce the Folding Tile Assembly Model (FTAM). The FTAM is intended to be a simplified mathematical model of self-assembling systems utilizing components which are able to dynamically reconfigure their relative 3-dimensional locations via folding and unfolding of flexible bonds between components. It is based on the abstract Tile Assembly Model (Winfree et al. 1998), and as such the fundamental components are 2-dimensional square tiles which bind to each other via glues on their edges. In contrast to the aTAM, in the FTAM each glue type can be specified to either form rigid bonds (which force two adjacent tiles bound by such a glue to remain fixed in co-planar positions) or flexible bonds (which allow two adjacent tiles bound by such a glue to possibly alternate between being in any of three relative orientations, as shown in Fig. 1). Because the FTAM is intended to be a simplified test bed for the study of flexible, reconfigurable self-assembling structures, in this paper we present a version of the model which makes many simplifying assumptions about allowable positions of tiles and dynamics of the self-assembly process, but which also differs greatly from previously studied self-assembling systems which allow reconfigurability (Hendricks et al. 2017; Jonoska and Karpenko 2014a, b; Fochtman et al. 2015; Padilla et al. 2014; Jonoska and McColm 2005, 2009; Aloupis et al. 2011; Aichholzer et al. 2017) as well as other computational studies of folding such as Aloupis et al. (2011) and Aichholzer et al. (2017).

In Sect. 2, we formally introduce the FTAM and provide definitions and algorithms describing its dynamics. The FTAM is designed to allow 2-D components to fold into 3-D structures, but with nondeterminism inherent in the flexibility of the bonds which make this possible since no single bond can lock two tiles rigidly into orthogonal planes. Instead, glues which can allow such positions are “floppy” and orthogonal tile orientations can only be fixed by leveraging additional, well positioned tile bindings. Therefore, in order to display how such flexibility can be effectively controlled to build designed structures, in Sect. 3 we provide a set of construction primitives and design principles which demonstrate how to control self-assembly in the FTAM. In Sect. 4 we present a pair of constructions which demonstrate the potential utility of reconfigurability of assemblies in the FTAM. In the first construction, an FTAM system, $T$ say, is given which produces a single terminal assembly that may be in many different configuration. In addition, a set, $S$ say, of $n$ distinct types of tiles are given such that for each subset of $S$, adding this subset of tiles of $S$ to the types of tile for $T$ gives a system with an assembly sequence that starts from the single terminal assembly of $T$ and yields a rigid terminal assembly (i.e., an assembly to which no tiles may bind and which, at a high-level, is in a configuration which cannot be folded via flexible glues to give another distinct configuration). Moreover, the resulting rigid assembly is distinct for each choice of subset of $S$. The second construction given in Sect. 4 demonstrates how a reconfigurable initial assembly can be transformed into either a volume-maximizing hollow cube or a small, tightly compressed brick by selecting between and adding one of two small subsets of tile types. These two constructions demonstrate how algorithmic self-assembling systems could be designed which efficiently (in terms of “input” specified by tile type additions) make drastic changes to their surface structures and volumes. These constructions show that FTAM systems can be designed which utilize reconfigurability. In Sect. 5 we show that this utility comes at a cost in terms of the computational complexity of determining some important properties of arbitrary FTAM systems. In particular, we show that, given an arbitrary FTAM system, the problem of determining whether or not it produces an assembly which cannot be reconfigured (via folding along tile edges bonded by flexible glues) is undecidable. Moreover, we show that, given an assembly, it is co-NP-complete to determine whether the assembly is rigid, i.e. has multiple valid configurations. Our final result modifies the previous to show that the problem of deciding if a given assembly for an FTAM system is terminal is also co-NP-complete. This is especially interesting since, in the aTAM, there is a simple polynomial time algorithm to determine if a given assembly is terminal.

2 Definition of the FTAM

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 2D unit square that can be translated, rotated, and reflected throughout 3-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 $i \in \{N, E, S, W\}$, which we refer to as $t_i$. Let $\Sigma$ be an alphabet of labels and $\Sigma_a = \{a^* | a \in \Sigma\}$ be the alphabet of complementary labels, then each side of each tile has a glue that consists of a label $\text{label}(t_i) \in \Sigma \cup \Sigma_a \cup \epsilon$ (where $\epsilon$ is the unique empty label for the null glue), a non-negative integer strength $\text{str}(t_i)$, and a boolean valued flexibility $\text{flx}(t_i)$. (See Fig. 1 for a depiction of the positions allowable by a flexible glue.)

A tile is an instance of a tile type. A placement of a tile $p = (l, n, o)$ consists of a location $l \in \mathbb{Z}^3$, a normal vector $n$ which starts at the center of the tile and points perpendicular to the plane in which the tile lies (i.e., $n \in \{+x, -x, +y, -y, +z, -z\}$), and an orientation $o$ which is a vector lying in the same plane as the tile which starts at the center of the tile and points to the $N$ side of the tile (i.e., $o \in \{+x, -x, +y, -y, +z, -z\}$). Note that by convention, to avoid duplicate location specifiers for a given tile, we restrict a location $l$ to refer to only the 3 possible tile locations with corners at $l$ and which extend in positive directions from $l$ along one of the planes (i.e., tiles are located by their vertices with the smallest coordinates). For any given $l$, there can only be a max of one tile with $n \in \{+x, -x\}$, one tile with $n \in \{+y, -y\}$, and one tile with $n \in \{+z, -z\}$.

Let $p = (l, n, o)$ and $p' = (l', n', o')$ be placements of tiles $t$ and $t'$, respectively, such that $p$ and $p'$ are non-overlapping and for some $i, j \in \{N, E, S, W\}$, sides $t_i$ and $t'_j$ are adjacent (i.e., touching). We say that $p$ and $p'$, have compatible normal vectors if and only if either (1) $n \equiv n'$, (2) $n$ and $n'$ intersect, or (3) $n$ and $n'$ intersect. We will refer to these three orientations as “Straight”, “Up”, and “Down”, respectively. Furthermore, if (1) label$(t_i)$ is complementary to label$(t'_j)$, (2) $\text{str}(t_i) = \text{str}(t'_j)$, (3) $\text{flx}(t_i) = \text{flx}(t'_j) = \text{False}$ and (4) $n$ and $n'$ are in a “Straight” orientation, then the glues on $t_i$ and $t'_j$ can bind with strength value $\text{str}(t_i)$ to form a rigid bond. Similarly, if (1) label$(t_i)$ is complementary to label$(t'_j)$, (2) $\text{str}(t_i) = \text{str}(t'_j)$, (3) $\text{flx}(t_i) = \text{True}$ and $\text{flx}(t'_j) = \text{True}$, and (4) $n$ and $n'$ are compatible, then the glues on $t_i$ and $t'_j$ can bind with strength value $\text{str}(t_i)$ to form a flexible bond.

1 We refer to the vectors \{(1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0), (0, 0, 1), (0, 0, -1)\} by the shorthand notation \{+x, -x, +y, -y, +z, -z\} throughout this paper.

2 The inverse function simply negates the signs of the non-zero components of a vector.

3 Note that any glue can only bind to a single other glue, and also that if edges of 4 tiles are all adjacent to each other, if glues of 2 tiles which are co-planar bind, then that “blocks” any possible binding between the other pair (which must be co-planar to each other) since that bond would have to cross through the existing bond.

We define an assembly $\alpha$ as a graph whose nodes, denoted $V(\alpha)$, are tiles and whose edges, denoted $E(\alpha)$, represent bound complementary glues between adjacent edges of two tiles. An edge between sides $i$ and $j$ of tiles $t$ and $t'$, respectively, is represented by the tuple $(t_i, t'_j)$, which specifies which sides of $t$ and $t'$ the bond is between. Whether or not it is flexible can be determined by $\text{flx}(t_i)$ and its strength can be obtained by $\text{str}(t_i)$ (since those values must be equal for both $t_i$ and $t'_j$).

We define a face to be a set of coplanar tiles that are all bound together through rigid bonds. Additionally, we define a face graph to be a graph minor 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 any single node in the first group of the assembly graph and any single node in the second group of the assembly graph. Conversely, the assembly graph is an inflation of the face graph.

An FTAM system is a triple $T = (T, \sigma, \tau)$ where $T$ is a finite set of tile types (i.e., tile set), $\sigma$ is an initial seed assembly, and $\tau$ is a positive integer which specifies the minimum binding threshold for tiles and is referred to as the temperature parameter. An assembly is $\tau$-stable if and only if every cut of edges of $\alpha$ which separates $\alpha$ into two or more components must cut edges whose strengths sum to $\geq \tau$. We will only consider assemblies which are $\tau$-stable (for a given $\tau$), and we use the term assembly to refer to a $\tau$-stable assembly.

Given an assembly $\alpha$, a configuration $\alpha_c$ is a mapping from every flexible bond in $\alpha$ to an orientation from \{“Up”, “Down”, “Straight”\}. An embedding $\epsilon_\alpha$ is a mapping from each tile in $\alpha$ to a placement. Given an assembly and a configuration, we can obtain an embedding by choosing any single 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. Note that, given tiles to which it is bound, their placements, and an orientation, there is only one tile location at which each additional tile can be placed. We say

![Fig. 2 Possible normal vectors of pairs of tiles. Those in a are compatible and allow a bond to form between complementary glues in the orientations “Up”, “Down”, and “Straight”, respectively. Those in b are not compatible.](image-url)
a configuration $c_x$ is valid if and only if an embedding obtained from the configuration (1) does not place more than one tile at any tile location, (2) doesn’t bond tiles through the same space, and (3) does not have contradicting bond loops. To elaborate on (2), while 4 glues can all be adjacent at one point, we allow them to bind in pairs in “Up” or “Down” orientations but do not allow both pairs to bind across the gap in “Straight” orientations. To elaborate on (3), 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. Examples of configurations that follow and contradict (3) are given in Fig. 3. Note that two embeddings that use different initial tiles and initial placements but the same configuration will be equivalent up to rotation and translation.

Let $x$ be an assembly and $c_x$ and $c'_x$ be valid configurations of $x$. If for every flexible bond $b \in x$ either $c_x(b) = Up$ and $c'_x(b) = Down$, or $c_x(b) = Down$ and $c'_x(b) = Up$, or $c_x(b) = Straight$ and $c'_x(b) = Straight$, we say that $c_x$ is the chiral configuration of $c'_x$ and vice versa. Note that the embeddings achieved from $c_x$ and $c'_x$ are reflections of each other. We refer to the special reconfiguration of an assembly to its chiral as inversion. We define a pattern of bond orientations, or simply just a pattern, to be a configuration and its chiral.

Given an assembly $x$ and two different embeddings $e_x$ and $e'_x$, we say that $e_x$ and $e'_x$ are equivalent, written $e_x \equiv e'_x$, if one can be rotated and/or translated into the other. If two embeddings are equivalent, this means they were computed from the same configuration, although possibly using a different placement for the initial tile.

We define the set of all valid configurations of $x$ as $C(x)$. We say that an assembly $x$ is rigid if (1) $|C(x)| = 1$, or (2) $|C(x)| = 2$ and the two valid configurations are chiral versions of each other. Conversely, if $x$ is not rigid, we say that it is flexible.

The frontier of a configuration $c_x$, denoted $\delta^T c_x$, is the set composed of all pairs $(t, B)$ where $t \in T$ is a tile type from tile set $T$ and $B$ is a set of up to 4 tile/glue pairs such that an embedding of $c_x$ would place each tile adjacent to one location such that a tile of type $t$ could bind to each glue for a collective strength greater than or equal to the temperature parameter $\tau$. Given an assembly $x$ and a set of valid configurations $C(x)$, we define the multiset of frontier locations of assembly $x$ across all valid configurations to be $\delta^T x = \bigcup_{c_x \in C(x)} \delta^T c_x$, i.e. $\delta^T x$ is multiset resulting from the union of the sets of frontier locations of all valid configurations of $x$.

Given assembly $x$ and valid configuration $c_x$, $\#(c_x)$ is the maximum number of new bonds which can be formed across adjacent tile edges in an embedding of $x$ which are not already bound in $x$ (i.e. these are tile edges which have been put into placements allowing bonding in configuration $c_x$ but whose bonds are not included in $x$). We then define $C_{\text{max}}(x) = \{ c_x | c_x \in C(x) \text{ and } \forall c'_x \in C(x), \#(c_x) \geq \#(c'_x) \}$. Namely, $C_{\text{max}}(x)$ is the set of valid configurations of $x$ in which the maximum number of bonds can be immediately formed.

Given an assembly $x$ in FTAM system $T$, a single step of the assembly process intuitively proceeds by first randomly selecting a frontier location from among all frontier locations over all valid configurations of $x$. Then, a tile is attached at that location to form a new assembly $x'$. Next, over all valid configurations of $x'$, a configuration is randomly selected in which the maximum number of additional new bonds can be formed (i.e. the addition of the new tile may allow for additional bonds to form in alternate configurations, and a configuration which maximizes these is chosen), and all possible new bonds are formed in that configuration, yielding assembly $x''$. Assuming that $x$ was not terminal and thus $x'' \neq x$, we denote the single-tile addition as $x \rightarrow_T x''$. To denote an arbitrary number of assembly steps, we use $x \rightarrow_{T_k} x''$. For an FTAM system $T = (T, \sigma, \tau)$, assembly begins from $\sigma$ and proceeds by adding a single tile at a time until the assembly is terminal (possibly in the limit). (See Algorithms 1 and 2 for pseudocode of the assembly algorithms.) For any $x'$ such that $\sigma \rightarrow_{T_k} x'$, we say that $x'$ is a producible assembly and we denote the set of producible assemblies as $\mathcal{A}[T]$. We denote the set of terminal assemblies as $\mathcal{A}_T[T]$.

Note that in this section we have provided what is intended to be an intuitively simple version of the FTAM in which the full spectrum of all possible configurations of an assembly are virtually explored at each step, and only those which maximize the number of bonds formed at every step are selected. Logically, this provides a model in which assemblies reconfigure into globally optimal configurations, in terms of bond formation, between each addition of a new tile. Clearly, depending on the size of an assembly and the degrees of freedom of various components afforded by flexible bonds, such optimal reconfiguration could...
conceivably be precluded by faster rates of tile attachments. Various parameters which seek to balance the amount of configuration-space exploration versus tile attachment rates have been developed to study more kinetically realistic dynamics, but are beyond the scope of this paper.

**Algorithm 1** A procedure to perform one step of the self-assembly process of FTAM system $T$

1. **procedure** ASSEMBLY-STEP($\alpha$, $T$) $\triangleright$ Takes an assembly $\alpha$ and FTAM system $T$
2. if $n(\alpha) = 0$ then
3. return $\alpha$ $\triangleright$ No frontier locations remain, $\alpha$ is terminal
4. else
5. Uniformly at random select $(t, B) \in \delta T \alpha$ $\triangleright$ Select a frontier location
6. Attach a tile of type $t$ with bonds to tiles in $B$, $\alpha \rightarrow T \alpha'$ $\triangleright$ Add a tile
7. Uniformly at random select $\alpha'' \in C_{max}(\alpha')$ $\triangleright$ Find new-bond-maximizing configuration
8. Form all bonds possible in $\alpha''$ to yield $\alpha'$ $\triangleright$ Form those bonds
9. return $\alpha''$ $\triangleright$ Return the new assembly
10. end if
11. end procedure

**Algorithm 2** A procedure to perform the self-assembly process of FTAM system $T$

1. **procedure** FULL-ASSEMBLY($\alpha$, $T$) $\triangleright$ Takes an assembly $\alpha$ and FTAM system $T$
2. $\alpha' = ASSEMBLY-STEP(\alpha, T)$
3. if $\alpha == \alpha'$ then
4. return $\alpha'$
5. else
6. return FULL-ASSEMBLY($\alpha'$, $T$)
7. end if
8. end procedure

### 3 Controlling flexibility to build structures

Our goal in this section to deterministically assemble certain shapes in the FTAM at temperature two. We define a shape to be a collection of connected tile locations. A shape is invariant through translation and rotation. Rather than go through an endless case-by-case analysis of all possible shapes, we focus on collections of 2D tile locations that form the outlines of three-dimensional shapes. We refer to these 3D shapes as polycubes and the set of 2D tile locations on their outer surface as an outline. We say that an FTAM system $T = (T, \sigma, \tau)$ deterministically assembles a shape $s$ if the embedding of all configurations $C_s$ of all terminal assemblies $A_{\square}[T]$ of the system $T$ have shape $s$.

Due to the definition of the model, the most prominent additional challenge that is present in FTAM systems over traditional 2D ATAM systems is controlling the orientation of different faces in the assembly relative to one another as the assembly process is occurring. In which case, the approach that we use to demonstrate shape building in the FTAM is to make an edge frame for each polycube using unique tile types and filling in each face. We define an edge frame to be the collection of the outer-most tiles of each face in the outline of a polycube. For now, we will make the assumption that every edge of the shape is connected and will address this later in the section. We claim that studying edge frames is sufficient for unveiling the power of the FTAM to orient new faces in the assembly process since, intuitively, the cooperation of other tiles on the edges of adjacent faces doesn’t provide additional help in correctly orienting those faces over just the tiles at the vertex. This intuition stems from the idea that the faces of a shape incident on a vertex interact on the same axes that the individual tiles incident on a vertex do.

One big deciding factor about whether the outline of a specific polycube can be made in the FTAM comes down to the types of vertices in that polycube. Because of this, we continue our analysis by breaking down the types of vertices that can exist on a polycube. Every type of vertex possible on a polycube can be enumerated by enumerating all polycubes that can fit inside a $2 \times 2 \times 2$ space that are distinct up to rotation and reflection. You can see the outcome of this enumeration in Fig. 4. In each polycube, the vertex type is illustrated at the center point of the $2 \times 2 \times 2$ space. The labels are to later reference each vertex type.

We categorize the types of vertices into two groups, simple and complex. In the enumeration in Fig. 4, the polycubes in the blue squares actually don’t have a vertex in the center. The vertices in the polycubes in red (1, 2, 5, 9) have three edges and three faces incident on the center point, creating what we refer to as a simple vertex. Of these 4 vertices, 1 and 9 are the same vertex type, which we will refer to as a convex vertex, and 2 and 5 are the same vertex type, which we will refer to as a concave vertex. The vertices in the polycubes in brown (3, 4, 6, 7) have more than three edges and more than three faces incident on the center point, creating what we refer to as a complex vertex. All of these complex vertices are unique, and we will refer to them by their number. The polycube in yellow (8) is a
special case in which there are more than three edges and more than three faces incident on the center point, but the polycube is arranged in a way that the center point can be thought of as two different simple convex vertices, one for each location that is missing a cube.

In addition to the vertex type, the system must also be able to deterministically assemble the vertex from the correct perspective. A perspective is the relative direction that the new edges of a vertex are pointing with respect to the tiles of the original edge. A vertex can be symmetric, meaning all edges have the same perspective, semi-symmetric, meaning some edges have the same perspective, or asymmetric, meaning no edges have the same perspective. Semi-symmetric and asymmetric vertices have to differentiate between the different perspectives the vertex can exist in. The simple convex vertex and vertex 3 are both symmetric, meaning they have only one perspective each. Vertex 4 and 7 are semi-symmetric, with vertex 4 having 2 perspectives (even though it has 4 edges) and vertex 7 having 3 perspectives (even though it has 6 edges). The simple concave vertex and vertex 6 and asymmetric, with the simple concave vertex having 3 perspectives (and 3 edges) and the vertex 6 having 5 perspectives (and 5 edges). An example of different perspectives can be seen in the difference between Fig. 5c, d. All together, there are 15 different perspectives, meaning we need 15 different tiling protocols to handle all situations.

We construct these protocols using (a) the number of tiles that are incident on the vertex that are bound in a loop (which we refer to as the loop length) and (b) the sequence of flexibility values in the bonds of the loop (which we refer to as the bond sequence). If a perspective has a unique protocol, then attaching a loop of tiles using the protocol will result in the only possible configuration available for the loop of tiles being the correct perspective. We will now look at the loop lengths and bond sequences of different vertices. For bond sequences, we will use the notation \((b, b, \ldots, b)\) where \(b \in \{R, F\}\) and \(R\) stands for rigid and \(F\) stands for flexible. The first bond in the sequence will be the edge assembling up to the vertex, and will therefore always be flexible, and the rest of the sequence will continue clockwise (for those vertices shown in Figures).

Fig. 5 a Original edge, b convex vertex, c concave vertex, from one unique perspective, d concave vertex, from another perspective

that the set of all bond sequences for a single vertex is a non-repeating cyclic permutation group.

We start with the simple vertices. First is the simple convex vertex. It has a loop length of 3 tiles and a bond sequence \((F, F, F)\). It is symmetric, meaning we don’t have to differentiate between the edges. Therefore, to make a simple convex vertex, an edge will just initiate a protocol where the last two tiles of the edge end in flexible glues and another tile that matches both glues attaches to complete the loop. The next vertex is the simple concave vertex. It has a loop length of 5 and can have a bond sequence of either \((F, R, R, F, F)\), \((F, F, R, R, F)\), or \((F, F, F, R, R)\). Using each bond orientation will result in a different perspective, meaning all 3 perspectives can be deterministically assembled. Therefore, attaching the loop of tiles with the first bond sequence will yield the vertex in Fig. 5d, the second bond sequence will yield the vertex in Fig. 5c, and the third sequence will yield the mirror opposite of that in Fig. 5d.

The most unique complex vertex is vertex 6. It has a loop length of 7 and can have a bond sequence of either \((F, R, F, R, F, F, F)\), \((F, F, R, F, R, F, F)\), \((F, F, F, R, F, F, F)\), \((F, F, F, F, R, F, F)\), \((F, F, F, F, F, R, F)\), or \((F, R, F, F, F, F, F)\). As with the simple concave vertex, each bond sequence results in a different perspective, allowing the system to differentiate between the 5 different perspectives of vertex 6. Similarly, vertex 4 also has its own unique combinations. It has a loop length of 6 and can have a bond sequence of \((F, R, F, F, F, R)\) or \((F, F, R, F, F, R)\). The two bond orientations correspond to the two perspectives of vertex 4, allowing both to be deterministically assembled.

Vertex 3 and 7, however, share a combination of a loop length and bond sequence. Both have a loop length of 6 and a bond sequence of \((F, F, F, F, F, F)\). Because of this, attaching a loop of 6 tiles using all flexible bonds at the end of an edge can result in either vertex. In addition, since flexible bonds can “mimic” rigid bonds using a “Straight” orientation, the loop of tiles can even configure into vertex 4. Note that the reverse is mitigated by the fact vertex 4 has rigid bonds and no bonds in vertex 3 or 7 are “Straight”. All together, it can end up in the one perspective from vertex 3, one of the two perspectives from vertex 4, or one of the three perspective from vertex 7. Given all these possibilities, vertex 3 and 7 cannot be deterministically assembled.

3.1 Assembly process

Now, we consider the assembly process. Let’s assume we start with a seed that is just the three tiles in a simple convex vertex. Notice that as the assembly process starts, the seed vertex and the edges that are growing out from it can invert as a whole but cannot otherwise
reconfigure (since that would require removing a bond from the assembly) (Fig. 6).

To grow edges, the system can use the following trivial protocol. Side 1 and side 2 refer to the two columns of tiles on the two faces that make up the edge. The hinge refers to the series of flexible bonds between tiles on side 1 and tiles on side 2.

1. An exposed rigid double strength glue on side 1 of the edge will attach a new tile \( t \) on side 1,
2. A flexible glue on tile \( t \) on side 1 and an exposed rigid glue on side 2 will cooperate to attach a new tile \( t' \) on side 2 of the edge,
3. A rigid double strength glue on tile \( t' \) on side 2 of the edge will attach a new tile \( t'' \) on side 2, and
4. A flexible glue on tile \( t'' \) on side 2 and a rigid glue on tile \( t \) on side 1 will cooperate to attach a new tile \( t'' \) on side 1 of the edge.

An edge can grow indefinitely by repeating this process using unique glues to grow up to a certain length. Notice that each new tile attaches using at least one rigid bond, meaning that, additional flexibility cannot be added to the edge past the flexibility of the hinge. Furthermore, there will only ever be one frontier location (per configuration, if multiple, but these are the same tile and bonds) on the edge at any assembly step, leaving no room for non-determinism.

Each time the assembly grows up to a vertex, it will attach the loop of tiles that make this new vertex. As long as the new vertex is not a reconfigurable vertex, it will be forced to take a configuration that agrees with configuration of the seed vertex. By this, we mean that, if the seed vertex were to invert at this point, the edge connecting the two vertices would invert, and the new vertex would therefore be forced to invert. This cause-effect relationship is true for any vertices (excluding reconfigurable vertices) connected by an edge, which means that, if any bond in the partial assembly were to reorient, the whole partial assembly must invert, i.e. inversion is the only possible reconfiguration. An example of an edge frame started from a potential seed is shown in Fig. 7.

We now prove a claim that assembling in the correct configuration or the chiral configuration is identical (since both configurations have the same frontier) and will therefore yield the same shape.

![Fig. 7 An assembling edge frame starting from a potential seed. Each edge grows up to a vertex and initializes other edges until the whole frame has filled out](image)

Claim 1. Every frontier location \( f \) in an assembly \( \alpha \) for a given configuration \( c_2 \) has a corresponding frontier location \( f' \) in \( \alpha \) in the chiral configuration \( c'_{2z} \), such that attaching \( f \) to \( \alpha \) in \( c_2 \) produces the same assembly but in the chiral configuration of attaching \( f' \) to \( \alpha \) in \( c'_{2z} \).

Proof. Notice that a frontier location in the FTAM is dependent on 12 neighboring tile locations, an “Up”, “Straight”, and “Down” location for each of the 4 sides of the tile. Also remember chiral configurations of an assembly \( \alpha \) produce embeddings of \( \alpha \) that are the reflections of each other. Now, take any frontier location \( f \) in \( c_2 \). By reflecting an embedding of \( c_2 \) over the plane that \( f \) exists in, the 12 tile locations that make \( f \) into a frontier location will still be neighboring \( f \), with the “Up” and “Down” neighboring locations switching places and also reflecting, thereby keeping the same glues incident on the location of \( f \). Since all the same glues are incident on the tile location, this location, which we will call \( f' \), is also a frontier location in \( c'_{2z} \) with the same tile type as in \( c_2 \), even if \( c'_{2z} \) includes some translation or rotation. By adding the tile to the assembly, since the frontier locations are on the plane of symmetry that we used to get the chiral configurations, adding it to the assembly in either configuration will produce two configurations that are also chiral configurations of each other.

As the edge frame grows, the system can use “filler tiles” to fill in the faces of the outline. On perfectly square faces, this can trivially be done, with filler tiles allowing to attach as the assembly grows. However, in cases where the face has a concave corner, a rectangular decomposition of the face with each rectangle being assigned a unique filler tile would prevent the filler tiles from overgrowing their bounds.

Once the assembly process has finished, the terminal assembly could also flip between the correct shape in its chiral. When there is at least one plane of symmetry in the shape, then reconfiguration in the assembly process actually will not prevent the system from being deterministic. This is because the chiral of a symmetric polycube is itself. Therefore, although the system will technically make two different terminal assemblies, one can be rotated into the other, meaning that the two different terminal assemblies
have the same shape by definition, making the system deterministic.

### 3.2 Multiple edge frames

Up to this point, we have assumed all the edges in a polycube are connected. However, this is not always the case. For example, anytime two pieces of a shape are connected by a set of coplanar tiles. Shapes like this are a problem because they require multiple edge frames to build, and similar to the chirality of asymmetric shapes, additional edge frames can also have chiral reconstructions. Therefore, disagreeing chiralities of the edge frames can configure the terminal assembly of a system into a shape that is neither the intended shape nor its chiral. In general, each additional edge frame doubles the number of configurations that the terminal assembly can exist in, only one of which (or two, if symmetric) is the desired shape. There are some exceptions to this such as blocking and symmetry.

Blocking refers to when the faces surrounded by one edge frame would collide with the faces of another if the chiralities of the edge frames disagreed. This is actually the case in the example given in Fig. 8a. In these situations, even if the additional edge frames are configured to the wrong chirality during the assembly process, eventually the tiles with the potential to collide will be added to the assembly and force the correct chirality of both edge frames with respect to each other. In Fig. 8c, you can see on the right how the inversion of the additional edge frame would cause the collision of tiles in the assembly. In this example, the yellow tile on the left of Fig. 8c would force the correct relative chirality.

One other aspect of a shape with multiple edge frames that may reduce the number of possible configurations it can exist in is, like with full shapes, symmetry of the edge frames. To utilize the same example, imagine if the shape in Fig. 8a did not contain the yellow tile on left end or the symmetric tile on the right end. In this case, the ends would be free to reconfigure into the wrong chirality. However, this doesn’t result in 4 different shapes that the mismatching chirality of 2 additional edge frames should produce. Instead, the inversion of the left end of the shape and not the right end yields the same shape as the inversion of the right end of the shape and not the left, resulting in only 3 different possible shapes.

Combining the results of this section, we get the following theorem.

**Theorem 1** A temperature two FTAM system can deterministically assemble the outline of any polycube that meets the following conditions:

1. the polycube is symmetric,
2. there are no reconfigurable vertices, and
3. the edges of the polycube are all connected.

### 4 Utilizing flexibility

As discussed previously, reconfigurability may be able to provide assembly systems with interesting properties that enable diverse applications. For example, changing geometry on the surface of a synthetic structure may allow it to interact with varying other structures in a system, or contracting/expanding volumes may impact how well it can diffuse through narrow channels. With a simple extension to the base FTAM model which allows an initial terminal assembly to form, and then at a later stage the addition of a new set of tile types allows the assembly to reconfigure, an assembly’s final shape can be locked in based on its final environment. As previously mentioned, we extend the FTAM here to allow such staged assembly as the simplest mechanism for leveraging this type of reconfigurability, but note that alternative mechanisms
could also work, such as glue activation and deactivation (Padilla et al. 2013).

4.1 Staged functional surface: maximizing the number of reconfigurations

For our first demonstration of a construction utilizing flexibility as a tool, we present a construction which maximizes the number of rigid configurations which a flexible assembly (formed during a first stage of assembly) can be locked into, based on the number of new tile types added during a second stage of assembly. Figure 9 gives a high-level schematic of a simple example of such a system. (Note that we omit full details of each tile type as these components can all be easily constructed using standard aTAM techniques and techniques from Sect. 3.) It shows the inner-makings of an initial structure that can later be modified by adding new tiles types into solution. We refer to this structure as a *film*. The film works by allowing the tiles in the very top layer to move freely. By adding select subsets of tile types during the second stage, prescribed tiles can be pinned up from the surface or pinned to the bottom layer of the assembly. Pinning up works by using the second layer of the film (from the bottom) to block the incoming tiles from folding down into the assembly, thereby forcing them to fold up. Pinning down works by connecting the top layer to the bottom layer of the film, forcing the tiles to fold down. The bumps formed from pinning up, also called *pixels*, can be arranged into a specified geometry, or *image*.

For this system, if the side lengths of the film are \( n \), note that there are \( O(n^2) \) potential pixel locations, meaning that there are a maximum of \( O(2^{n^2}) \) possible pixel configurations (i.e. each can be either up or down in any given configuration). To transform the flexible film into a rigid configuration with a particular set of pixels projecting upward, it is necessary to add tiles of \( O(n^2) \) tile types corresponding to the up or down orientations, which is optimal as each tile type is encoded by a constant number of bits and \( \log(O(2^{n^2})) = O(n^2) \) bits are necessary to uniquely identify each of the \( O(2^{n^2}) \) configurations. Note that although these reconfigurations are relatively trivial, the differences in the sizes of the reconfigurable sections can be arbitrarily large without requiring more unique tile types to be added in the second stage. The goal of this construction is to display a maximum number of resulting rigid configurations from an optimal number of additional tile types in the second stage.

4.2 Compressing/expanding structures

We now demonstrate a construction which is able to take advantage of the flexibility of bonds in the FTAM to allow a base assembly to lock into an expansive, rigid but hollow configuration given the addition of one subset of tile types in the second stage, or to instead lock into a compressed, compact and dense configuration given the addition of a different subset of tile types.

Figure 10 shows the transition of one end state to another. Starting with state (F), this general assembly can be assembled using \( O(\log(n)) \) tile types. To transition to state (A), tile types are added that attach around the perimeter of the shape and cause it to configure into a cube of \( n \times n \times n \) dimensions. To transition to state (K), tile types are added that form a system of “tabs” and “caps” that compress the assembly into a brick of \( O(n) \times \frac{n}{\sqrt{n}} \times \frac{n}{\sqrt{n}} \) configuration.
$O(\sqrt{n}) \times O(\sqrt{n})$ dimensions. Both of these transitions only require a constant number of additional tile types.

5 Complexity of FTAM properties

In this section we consider the computational complexity of determining interesting properties about FTAM systems and the assemblies within them.

5.1 Determining if a system produces a rigid assembly is uncomputable

**Problem 1** (Rigidity-from-system) Given an FTAM system $T$, does there exist an assembly $\alpha \in \mathcal{A}[T]$ such that $\alpha$ is rigid?

**Theorem 2** Rigidity-from-system is undecidable.

First, we consider the general structure of a commonly used type of aTAM tile assembly system for simulating the behavior of Turing machines. A zig-zag aTAM system is one which grows in a strict row-by-row ordering. More specifically, the first row grows either left-to-right or right-to-left, completely, at which point the second row begins growth in the opposite direction. When it completes, the third grows, again in reversed direction, and so on. Given a Turing machine $M$, $M$ can be simulated by a temperature-2 zig-zag aTAM system, $\mathcal{P}$ say, such that if $M$ halts, a final “halting” tile attaches to the westernmost column in the northernmost row of the terminal assembly of $\mathcal{P}$. One can also show that $M$ can be simulated by a zig-zag system which produces a single terminal assembly such that the westernmost tiles of this assembly (possibly including the halting tile) are colinear. Moreover, such an aTAM system gives rise to an FTAM system, $\mathcal{T}$ say, where the tile types of $\mathcal{T}$ are identical to the tile types of the aTAM system and all glues are rigid. To show Theorem 2, we consider the FTAM system $\mathcal{T}'$ that is obtained from $\mathcal{T}$ by (1) modifying the glues on north and south edges so that they are flexible, and (2) adding appropriate glues to tile types and tiles to the tile set of $\mathcal{T}$ so that if $M$ halts, tiles of these types initially bind to the west edge of the halting tile (via an added strength-2 glue) of the terminal assembly of $\mathcal{T}'$, and then cooperatively bind one at a time along the west edges of the westernmost tiles of this terminal assembly (via strength-1 glues added to these west edges and north/south glues of the tiles of the additional tile types) to form a single tile wide column of tiles each of which is bound to a westernmost tile in the terminal assembly of $\mathcal{T}'$. Moreover, the north and south glues of the tile types that bind to form the column of westernmost tiles are rigid. We call such a column of tiles a “backbone”. Then, as the east/west glues of tiles belonging to any row of tiles in the terminal assembly of $\mathcal{T}'$ are rigid, and since a backbone of tiles self-assembles iff $M$ halts, we see that the terminal assembly of $\mathcal{T}'$ is rigid iff $M$ halts. In other words, we have a system such that for any terminal assembly $\alpha$, $\alpha$ is rigid iff $M$ halts. See Fig. 11 for an intuitive description of self-assembly in $\mathcal{T}'$ for a simulation of a machine that halts. This suffices to show Theorem 2.

5.2 Determining the rigidity of an assembly is co-NP-complete

**Problem 2** (Rigidity-from-assembly) Given an FTAM system $\mathcal{T}$ and assembly $\alpha \in \mathcal{A}[\mathcal{T}]$, is $\alpha$ rigid?

**Theorem 3** Rigidity-from-assembly is co-NP-complete.

To prove Theorem 3, we prove the following lemmas.

**Lemma 1** Rigidity-from-assembly is in co-NP.

**Proof** To illustrate this, we take an instance of the problem that contains the FTAM system $\mathcal{T}$ and assembly $\alpha \in \mathcal{A}[\mathcal{T}]$. Our certificate in this instance will be configurations $c_\alpha$ and $c'_\alpha$. Since a configuration is simply a mapping from every flexible bond in $\alpha$ to an orientation, each configuration requires $O(|\alpha|)$ space, and thus the certificate is polynomial in the size of $\alpha$. To determine if the certificate is valid, and thus if $\alpha$ is flexible (and therefore not rigid), we first check that $c_\alpha$ and $c'_\alpha$ are valid encodings of a configurations, meaning they each map every flexible bond in $\alpha$ to an orientation from $\{"Up", "Down", "Straight"\}$. Then we must ensure that $c'_\alpha$ is different than $c_\alpha$. Both of
these can be done in linear time with respect to the number of flexible bonds in the assembly. Next, we compute embeddings of \( z \) from \( c_z \) and \( c'_z \), taking linear time in the number of tiles in the assembly. While computing the embeddings, we simply check that no tile is assigned a placement already taken by another tile, that no bonds overlap the same space, and that every tile is adjacent to the tiles it is connected to in \( z \) such that their glues line up correctly. Computing the embeddings and checking these conditions takes linear time with respect to the number of tiles in the assembly. While computing the embeddings and checking these conditions takes linear time with respect to the number of tiles in the assembly. If all of these conditions are met, then both \( c_z \) and \( c'_z \) are valid configurations of \( z \), and therefore \( z \) is not rigid. Since the certificate has polynomial size in relation to \( z \) and can be verified in polynomial time to show that \( z \) is not rigid, determining if \( z \) is rigid is in co-NP. □

**Lemma 2**  The complement of rigidity-from-assembly is NP-hard.

To prove Lemma 2, we will reduce 3SAT to the complement of the rigidity-from-assembly problem. For our reduction, we introduce a new construction that we will subsequently refer to as the 3SAT machine. This is a computable assembly that is made up of four modules that connect together and will be able to reconfigure if and only if the corresponding 3SAT formula has a satisfying assignment. The four modules are the evaluation space (ES), the variable constraint gadgets (VCG’s), the trivial assignment hat (TAH), and the satisfying assignment hat (SAH).

Before we get into the details, we define a few terms and prove a few lemmas that we will use as tools in the overall proof. We define entanglement to be the cause-effect relationship of different flexible bonds in one set of faces in which the inversion of one flexible bond causes the inversion of all other flexible bonds. A set of faces that have been entangled form a rigid component.

We define a traditional 4-sided loop, as a cycle of faces in which each face is bound on opposite ends to two other faces. In other words, both pairs of faces that are opposite of each other will exist in planes in the same orientation \{XY, YZ, ZX\}. Notice that a traditional 4-sided loop will always be a rigid component. The four flexible bonds between the four faces that make up the loop will always have to have either an “UUUU” or “DDDD” configuration.

**Claim 2**  Given an assembly \( z \) and a configuration \( c_z \), for any rigid component \( \text{comp}_{\text{rigid}} \) in \( z \), if \( z \) has another valid, non-trivial configuration \( c'_z \) that it can exist in, then there must also exist another valid, non-trivial configuration \( c''_z \), where \( c'_z \) may or may not equal \( c''_z \), in which the rigid component \( \text{comp}_{\text{rigid}} \) is not inverted.

**Proof**  Assume that in \( c'_z \), \( \text{comp}_{\text{rigid}} \) is not inverted. Then \( c'_z = c''_z \). However, if we assume that in \( c'_z \), \( \text{comp}_{\text{rigid}} \) is inverted, then \( c'_z = \text{inverse}(c''_z) \). □

**Claim 3**  As a corollary of claim 2, for any rigid component \( \text{comp}_{\text{rigid}} \) in \( z \), if \( z \) in \( c_z \) does not have another valid configuration with \( \text{comp}_{\text{rigid}} \) in the same orientation, \( z \) itself is a rigid component.

These claims give us a powerful tool in proving which pieces of an assembly are rigid components, since it allows us to assume one smaller rigid component will not reorient and then examine if any other pieces can reorient in relation to it.

**Claim 4**  If two rigid sections \( \text{comp}_{\text{rigid}}^1 \) and \( \text{comp}_{\text{rigid}}^2 \) share two faces \( p_1 \) and \( p_2 \) that are not coplanar, then \( \text{comp}_{\text{rigid}}^1 \) and \( \text{comp}_{\text{rigid}}^2 \) can be entangled into one rigid component.

**Proof**  Proof by contradiction. Assume \( \text{comp}_{\text{rigid}}^1 \) can reorient relative to \( \text{comp}_{\text{rigid}}^2 \). Since \( \text{comp}_{\text{rigid}}^1 \) is rigid in and of itself, the only way it can reorient is to invert. Now, assume the inversion happens over \( p_1 \). In this case, \( p_2 \) must now be flipped over the plane that \( p_1 \) exists in. Since \( p_2 \) is also part of \( \text{comp}_{\text{rigid}}^2 \) and is in a new location, \( \text{comp}_{\text{rigid}}^2 \) must also have reoriented to avoid breaking bonds with \( p_2 \), the only way to do which would be inverting. Now, in the case that the inversion happens over another face, the plane that contains that planar section cannot contain both \( p_1 \) and \( p_2 \) (since they are not coplanar by the claim) and therefore at least one of the shared planar sections will be in a new location. This means that \( \text{comp}_{\text{rigid}}^2 \) still had to reorient. Since both of these cases lead to contradictions, \( \text{comp}_{\text{rigid}}^1 \) and \( \text{comp}_{\text{rigid}}^2 \) must be part of the same rigid component. □

This provides us another powerful tool, showing that two shared non-coplanar faces is all that is required to entangle two rigid components into one.

Now that we have some tools to use in our proof, we give a proof overview in two main steps. The first step will be to take some of the pieces in the 3SAT machine and entangle them together into larger rigid components. Figure 12 shows how these rigid components will be made. The second step will be to show that the remaining pieces interact in such a way that the rigidity of the machine is indeed linked to the underlying 3SAT problem.

### 5.2.1 Evaluation space (ES)

The evaluation space, shown in Fig. 13, is a frame that facilitates interactions between the variable constraint gadgets, trivial assignment hat, and satisfying assignment...
hat. The frame consists of three evaluation gadgets per row (which we colloquially referred to as “bumps” in the main section), with as many rows as clauses in the corresponding 3SAT instance. The evaluation gadget consist of three tiles that form a bump over or under the open holes in the evaluation space. The three tiles that compose the evaluation gadget consist of two tiles in the ZX plane and a tile in the XY plane connecting them. An evaluation gadget forced up (Fig. 13a) represents a literal evaluated to “True”, and an evaluation gadget forced down (Fig. 13b) represents a literal evaluated to “False”. In the situation where the corresponding 3SAT instance does not have a satisfying assignment, the evaluation gadgets will collide with tiles from the satisfying assignment hat, preventing any configuration other than the initial. Every variable in the 3SAT instance has a corresponding evaluation gadget unless that variable has only positive literals or negative literals. If this is the case, we exclude that particular evaluation gadget since all instances of that variable can be assumed to be true, so there is no reason to collide the satisfying assignment hat.

Attached to the evaluation space on opposite sides are the satisfying assignment hat and the trivial assignment hat. These pieces are attached to the evaluation space in such a way that exactly one of the them will be pressed against the evaluation space in any possible configuration. The satisfying assignment hat can only reorient down onto the evaluation space, as shown in Sect. 5.2.4, if and only if there is a solution to the corresponding 3SAT instance; otherwise, the configuration where the trivial assignment hat is pressed against the evaluation space, as shown in Fig. 14a, will be the only valid configuration that the assembly can exist in.

5.2.2 Variable constraint gadgets (VCG’s)

The variable constraint gadgets sit below the evaluation space and interact with the evaluation gadgets to ensure that each instance of a variable, even the negated instances, agree with each other. Only variables that have at least one positive literal and at least one negative literal will have a variable constraint gadget associated with that specific variable. Since other variables with all positive or all
negative instances have no evaluation gadgets, they also do not have variable constraint gadgets.

The variable constraint gadgets are designed such that all eligible variables are assigned a unique level at a different Z value below the evaluation space. Each positive/negative instance of a single variable has two parallel strings of tiles called chimneys that connect the XY tile of all evaluation gadgets that correspond to that positive/negative variable to the variable constraint level (VCL) of that positive/negative variable. For example, all evaluation gadgets for \( x_1 \) literals have chimneys that extend to the same variable constraint level at a specified Z value, as well as all evaluation gadgets for \( \overline{x}_1 \) literals.

The variable constraint levels, shown in Fig. 15, connect all instances of a variable. There are two levels per variable, one for all positive literals, i.e. \( x_1 \), and one for all negative literals, i.e. \( \overline{x}_1 \). The two levels are then connected by a bridge, which connects to the end of both crossbars as seen in the Figure. The bridge is simply domino that connects the two levels in the ZX plane. Its purpose is to ensure that the levels exist in XY planes that are two units in the Z direction apart. Since the levels are connected to the chimneys and to the respective evaluation gadgets, we will show that this ensures that the evaluation gadgets for positive and negative literals of the same variable always disagree.

Claim 5  The variable constraint gadgets can only reorient by changing truth value or being inverted.

Proof  To recap, for any variable \( x \) with \( p \geq 1 \) instances of the positive literal and \( n \geq 1 \) instances of the negated literal, the variable constraint gadget is made up two levels (one for \( x \) and one for \( \overline{x} \)), \( p \) pairs of chimneys to one level, connected to the evaluation space collectively by \( p \) evaluation gadgets, \( n \) pairs chimneys to the other level, connected to the evaluation space collectively by \( n \) evaluation gadgets, and one bridge between the two levels. Evaluation gadgets are made up of three pieces, one in the XY plane and two in the ZX plane.

First, notice that an XY piece of an evaluation gadget, a pair of chimneys, and the level they are connected to make a 4 sided loop. Take anyone of these loops and assume it is rigid by claim 3. Obviously, the orientation of every other chimney loop that exists between the evaluation space and that specific level must also remain unchanged in order for them to connect to the level. Now, we have a rigid component of one level, all the chimneys attached to it, and XY pieces of their respective evaluation gadgets. This rigid component can move up or down by two units due to reorientation in the ZX pieces of the evaluation gadgets. We apply the same argument to the level, chimneys, and evaluation gadgets corresponding to \( \overline{x} \) literals.

Now, our variable constraint gadget for \( x \) consists of two entangled rigid components (which we will now call \( l \) and \( r \)), two sets of ZX pieces from the evaluation gadgets (one connected to \( l \) and one connected and \( r \), and the bridge. We already know that a reorientation can happen that moves either \( l \) or \( r \) up two units in the Z direction, the other down two units in the Z direction, and causes a transformation of all ZX pieces from the evaluation gadgets and the bridge. However, this is the only reorientation that can happen. To show this, notice that any pair of ZX pieces for the evaluation gadgets must agree to be connected to the evaluation space through both \( U \) bonds or both \( D \) bonds in order to have a gap of one between them for the XY piece to fit into. Since the rigid component \( l \) or \( r \) has already been shown to be rigid from one ZX evaluation piece to another, the orientation of this pair of ZX evaluation pieces will force the orientation of all other pair of evaluation pieces also connected to the \( l \) or \( r \) piece. The same argument goes with the ZX evaluation pieces connected to the other \( l \) or \( r \) rigid component. Notice that the \( l \) and \( r \) component can’t be on the same level, otherwise the two ends connected by the bridge would be incident on each other, leaving no
room for the bridge itself. Therefore, one rigid component (l or r) must have each of its ZX evaluation pieces be up, while the other rigid component must have each of it ZX evaluation pieces be down. Therefore, the only ambiguity in the configuration of the variable constraint gadget (other than the trivial ambiguity of inversion) is determining which rigid component between l and r is up and which rigid component is down. Because all variable constraint gadgets have the same structure, this argument holds for the variable constraint gadget corresponding to all variables in the formula.

5.2.3 Trivial assignment hat (TAH)

The purpose of the trivial assignment hat is to prevent the reorientation of any piece in the assembly as long as it is pressed against the evaluation space. It does this in two ways. First, it has what we labeled as “Force Bumps” that force the orientation of all evaluation gadgets and variable constraint gadgets, as shown in Fig. 16a. These are designed such that all negative literals in the evaluation space have a corresponding force bump on the trivial assignment hat and all positive literals do not. This effectively assigns a value of “false” to all variables in the formula, hence the name “trivial assignment hat”, forcing the evaluation gadgets and variable constraint levels of all negative variables down and thereby forcing the evaluation gadgets and variable constraint levels of all positive variables up.

The second way the trivial assignment hat disables reorientation is by having an additional piece that blocks otherwise-free pieces in the satisfying assignment hat to move. This piece can be seen in Fig. 16b. It will later be proven that this is sufficient to show that the machine is rigid in the case when no satisfying assignment exists to the corresponding 3SAT problem.

Claim 6 The trivial assignment hat can’t reorient without being inverted.

Proof To clarify some of the terminology of this section refer to Fig. 12 or 17a.

Here we will show that most faces in the trivial assignment hat are part of a traditional four sided loop utilizing the “Left Side” piece and one of the two “Right Side” pieces, i.e. every loop will look like: Left Side, piece one, Right Side, piece two, Left Side. The planar sections that will be substituted into “piece one” and “piece two” are as follows: (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). By claim 4, since each loop contains Left Side and one of the two Right Side’s, we are left with two rigid components (corresponding to the split up of the Right Side’s). However, again by 4, both rigid components share Left Side, Main Top, and Main Bottom, none of which are coplanar, and can therefore be entangled into one rigid component. The only remaining planar sections in the trivial assignment hat are the “Force Bumps”, each of which forms a traditional four sided loop with the “Main Bottom”. Since each of these loops is rigid, the piece that is opposite of the “Main Bottom” must be in the same orientation as “Main Bottom” but in a plane that is one unit away in either normal direction. Since the “Main Top” is in the same orientation and in the plane that is one unit above the “Main Bottom”, no force bump can invert because it would collide with the “Main Top”. Therefore, each force bump can be entangled into the rigid component, leaving us with one rigid trivial assignment hat. □
5.2.4 Satisfying assignment hat (SAH)

The satisfying assignment hat comes down and presses up against the evaluation space to make an additional valid configuration for the whole machine if and only if there is a solution to the inputted 3SAT problem. The satisfying assignment hat works by using a checker for each clause in the formula. The checker is encased in a structure, shown in Fig. 18, that allows it to be in three different orientations, shown in Fig. 19. These different orientations each project a blocking tile, shown in yellow, to a different location to block one of the three variables in the clause. The checkers can only be in three different orientations due to the structure the checker is encased in. In Fig. 18 we can see this illustrated by the rigid outer structure and the partly flexible checker inside the structure. Marked in yellow, the flexible points in the checker allow for the checker to only “fold into” one of the available, raised parts of the structure. If the checker were to fold into a non-raised portion of the structure, the tiles would simply conflict with the outer structure, shown in red in Fig. 18. If the satisfying assignment hat is able to press against the evaluation space, that means the checker for each clause has found at least one of the variables to be true in each clause, allowing the checker to be in the orientation that projects the blocking tile over the evaluation gadget that corresponds to that variable.

**Claim 7** The satisfying assignment hat can’t reorient without being inverted.

**Proof** (To clarify some of the terminology of this section refer to Fig. 12 or 17b.) First, notice the traditional 4-sided loop between the Tunnel Bottom and Tunnel Top utilizing the Tunnel Top Side Support’s. Similarly, the Tunnel Bottom and Connection 2 piece form a 4-sided loop. We can entangle these two loops because, if one were to invert without the other, the three Tunnel Slack 2 pieces would have to collide with the Tunnel Bottom piece in order to connect the two. Since the Connection 2 piece and the Tunnel Top are part of the same rigid component, we can also entangle the three Tunnel Slack 2 pieces. Now, notice that the Tunnel Bottom, Tunnel Back, and any Left Starter or Right Starter piece form a vertex. This means that, in relation to the Tunnel Bottom, the Tunnel Back must either be Up or Down but cannot be straight. However, if it were to be Down (thereby inverted relative to the rigid component), its connection to the Tunnel Back would be 4 down in the Z dimension and 4 to the left in the X dimension to the connection between the Tunnel Slack Front 1 and the Connection 2. Since this Manhattan distance is 8 and the combined length of the four piece that need to make up this distance (Connection 1, Tunnel Slack Back 1, Tunnel Slack Top 1, and Tunnel Slack Front 1) is also 8, the pieces would have to form a shortest path from one connection to the other. However, since every shortest path would intersect with either the Tunnel Bottom or Tunnel Back, it cannot make this distance. Therefore, the Tunnel Back (and Left Starters, Right Starters, and First Sectionals) can be entangled into the rigid component. Now, the Manhattan distance that needs to be made up by the 4 pieces is just 4. Since the respective lengths of the 4 pieces in the order previously given are 1, 2, 3, 2, the only arrangement that would give a total displacement of 4 would for the 2 length pieces (Tunnel Slack Back 1 and Tunnel Slack Front 1) to cancel out and for the other two pieces (Connection 1 and Tunnel Slack Top 1) to make up the distance. This leaves two options, the back and front slack folded away from the Tunnel Bottom or towards it. As mentioned earlier, if the slacks fold towards the Tunnel Bottom, the Tunnel Slack Top 1 would collide with the Tunnel Bottom. Thus, there’s only one configuration of these pieces with respect to the rigid component, meaning they can be entangled, leaving one big rigid component.

5.2.5 3SAT checker machine

There is one last piece of the assembly we must introduce, which we refer to as the rope. This is a $15 \times 1$ face of tiles that connects the trivial and satisfying assignment hats. It connects to the “Tunnel Top” on the SAH and the “Main Top” on the TAH. It’s purpose is to prevent the invalid configuration from Fig. 14d where neither hat is pressed against the evaluation space.
Now that we have introduced all the pieces of the assembly, we need to focus on how the remaining flexible bonds can move relative to one another. We group the remaining flexible bonds into three groups: the main loop bonds, the checker bonds, and the variable constraint bonds. The main loop group consists of four bonds: the ES to the SAH, the SAH to the rope, the rope to the TAH, and the TAH to the ES. Notice that these bonds form a loop. By looking at Fig. 20, you will notice this loop can only take two configurations, “UDSS” and “SSDU” (with respect to ES-SAH, SAH-rope, rope-TAH, TAH-ES, with the normal of the rope pointed in the +Z dimension and the normal of the ES pointed in the –Z dimension). We refer to these two configurations as the trivial state and satisfied state, respectively.

The checker group consists of 16 bonds for each checker gadget in the SAH. These 16 bonds can be viewed as two subgroups of 8 which must be the same configuration from the set of “UDDUUDDU”, “UDDUSSSS”, and “SSSSSSSS” (with respect to the sequences of bonds starting at the “First Sectional” piece and ending with the “Third Sectional and Blocker” piece, with the normal of these two pieces pointed at the “Tunnel Top”). The sequences for the two subgroups of any checker gadget must match since both sequences of bonds end with the “Third Sectional and Blocker” piece, which connects with both previous pieces at the same X coordinate. As proof, we direct the reader to Fig. 18 to see the reasoning that these are the only three possible configurations. Note that the invalid configuration in the figure would be “SSSUUDDU”.

The variable constraint group consists of all the bonds between the ES and the ZX evaluation bump pieces, all the bonds between the ZX evaluation bump pieces and the XY evaluation bump pieces, and all the bonds between the VCL’s and the bridges. As we saw in claim 5, this entire group of bonds can only have as many possible configurations as there are possible assignments to the variables in the corresponding boolean formula, i.e. \(2^n\) where \(n\) is the number of variable in the formula. An intuitive explanation of this is that each variable has a variable constraint gadget, each of which can only take two states, one where the variable is false and one where the variable is true.

Finally, we show what can happen when the main loop is in the trivial state and satisfied state. The last two proofs are the same as the proofs in the main section but go into more detail.

**Claim 8** Neither the trivial nor the satisfying assignment hat can invert without the other.

**Proof** Notice that the displacement vectors in Fig. 20 between the bond where either hat connects to the ES and the bond where the same hat connects to the rope is some rotation of the vector \((2, 3)\) in the ZX plane (it is the reverse of the apparent vector in the Figure, since the illustration is from the –Y perspective instead of the +Y perspective). If one hat were to invert without the other, the displacement vector of the inverted hat would change to some rotation of the vector \((3, 2)\). Then, for the two displacement vectors to have the same Z value (to have the ends of the rope be in the same Z plane) the X values must be the negation of each other. This leaves the possible distance between the two bonds at the ends of the rope to be 9, 11, 19, or 21. These numbers were computed 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 the displacement vectors must point in opposite X directions. Since none of these possible distances are 15, the actual distance of the rope, then none of the possible configurations resulting from an inversion of one hat without the other are valid.

**Claim 9** If the main loop is in the trivial state, no other bond can reconfigure.

![Fig. 20](image) A view of the main loop of the assembly from the negative Y direction. The arrows represent the displacement between the bonds from the evaluation space to each hat and the bonds from each hat to the rope. The orientation labels are for the bonds between the evaluation space and each hat. The illustration is meant to show that the only positions that the two hats can be in such that their bonds to the rope are in the same Y plane and exactly 15 units apart in the X plane are the two labeled positions for the rope.
Proof First, we also prove that the variable constraint gadgets can’t invert without the SAH and TAH. This is easy to see from Figs. 14 and 21. Since the TAH lays across the entire ES, if the variable constrain gadgets were to invert without the other main components, the chimneys would collide with the “Main Bottom” and “Main Top” pieces in the TAH. Therefore, the variable constraint gadgets can’t reorient without both of the other main components reorienting as well.

Now, we show the remaining flexible bonds, the checker group and the variable constraint group, can’t reorient either. In the trivial state, the checkers in the SAH are blocked by the extension of the TAH, which prevents the bonds from taking any configuration other than “UDDUDDU”. The bonds in the variable constraint group are also forced into a configuration by the TAH, since the force bumps on the TAH force all evaluation gadgets for positive literals to be popped down, thereby forcing their respective VCL’s down, thereby forcing their opposing VCL’s up, and thereby forcing all evaluation gadgets for negative literals to be popped up. Therefore, if the main loop bonds are fixed in the trivial state, the entire assembly is rigid.

Claim 10 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 First, we address the inversion of variable constraint gadgets. Since the TAH is no longer pressed against the ES, the variable constraint gadgets could invert across the XY Evaluation Bump Pieces that that specific variable constraint gadget is bound to. However, this inversion would not affect the functionality of the variable constraint gadget, since it would still cause all positive literals of one variable to be one truth value and all negative literals of that variable to be the other truth value. Therefore, we can ignore any potential inversion of these gadgets in this proof. For elegance, the “Tunnel Top” piece of the SAH could be extended such that it blocks these inversions.

For the forward direction of the bijection, we prove that, “If the assembly configures such that no tiles overlap, there is a satisfying assignment”. We know that, in the satisfied state, the “Third Sectional and Blocker” face in each checker gadget of the SAH must occupy the tile location that is one unit in the Z dimension above one of the three evaluation gadgets in the clause that the checker gadget corresponds to. Therefore, if no tiles overlap, one of the literals in that clause must have evaluated to true so that it is popped down in the Z dimension, allowing the checker gadget to occupy the space above. If this evaluation gadget represents a positive literal, we can start building the satisfying assignment by setting that variable to true. If the evaluation gadget represents a negative literal, we set the variable to false. We do this for each checker gadget. We know that we will never have to reassign a variable a new value, since the variable constraint gadgets ensure that every literal of the same type agrees with each other. After we go through each checker gadget, we have a variable assignment that satisfies every clause in the boolean formula. Any variable that hasn’t been assigned can be given either truth value.

For the reverse direction, we prove that, “If there is a satisfying assignment, the assembly can configure such that no tiles overlap”. Here, we can take every variable in the satisfying assignment, and if it is assigned the value true, we raise the VCL corresponding to the negative literals of that variable (and pop up the connected evaluation gadgets) and lower the VCL corresponding to the positive literals of that variable (and pop down the connected evaluation gadgets). If the variable is assigned the value false, we do the opposite. Because this is a satisfying assignment, at least one evaluation gadget in each clause must be popped down, allowing each checker gadget to have one configuration that doesn’t overlap with an evaluation gadget.

To prove Lemma 2, we use the previous claims as follows. Take the 3SAT machine in the trivial state. By claim 10, if the corresponding 3SAT formula has a satisfying
assignment, then the machine has at least one additional configuration in the satisfied state it can reconfigure into. If the corresponding 3SAT formula does not have a satisfying assignment, there are no valid configurations in the satisfied state. By claim 9, there is only one configuration in the trivial state. Therefore, determining if the assembly has multiple valid configurations, the complement of rigidity-from-assembly, is polynomial time reducible from 3SAT and NP-hard.

Finally, Theorem 3 is proven by Lemmas 1 and 2.

5.3 Determining the terminality of an assembly is co-NP-complete

In addition to rigidity, terminality is another useful-to-know property of assemblies. Using much of the same logic from the previous result, we can prove a similar result regarding the terminality of arbitrary assemblies.

**Problem 3** (Terminality-from-assembly) Given FTAM system $T$ and $x \in A[T]$, is $x$ terminal?

**Theorem 4** Terminality-From-Assembly is co-NP-complete.

We prove Theorem 4 via the following two lemmas.

**Lemma 3** The complement of terminality-from-assembly is in NP.

**Proof** For an instance of the problem, we are given an FTAM system $T = (T, \sigma, \tau)$ and assembly $x$. Our certificate in this case includes a configuration $c_x$ for the assembly $x$ and a frontier location $f$. Similar to in the proof of Lemma 1, (and since the encoding of $f$ requires space $\leq |x|$) we know that the certificate is polynomial in size to $x$. Also, we can check the validity of configuration $c_x$ in polynomial time. Now, we simply need to verify the frontier location $f$ (a) isn’t already occupied by a tile and (b) is adjacent to tiles in $x$ while it’s in configuration $c_x$ such that the adjacent glues allow the tile specified by $f$ to bind to $x$ with bonds collectively $\geq \tau$ strength, which can be done in time $O(|T|) + O(|x|)$. As the certificate has polynomial size in relation to $x$ and can be verified in polynomial time to show that $x$ is not terminal, determining if $x$ is terminal is in co-NP.

**Lemma 4** The complement of terminality-from-assembly is NP-hard.

To prove Lemma 4, we use almost identical techniques as for the proof of Lemma 2, with a slight modification to the 3SAT machine used to prove Lemma 4. This assembly has a bond between a piece called the “Rope” and the TAH that is in a “Straight” orientation when the machine is in trivial state and in a “Down” orientation in the satisfied state. For both tiles that make up this bond, add a unique flexible glue that is on the side 90 degrees clockwise from the original bond on one tile and 90 degrees counter-clockwise from the original bond on the other tile. This way, both unique flexible glues are pointed in the same dimension. In the trivial state, these glues are adjacent to two different tile locations that are adjacent themselves. However, if the assembly can reconfigure into the satisfied state, these glues become adjacent to a mutual tile location. Therefore, adding a tile type to the system with the complements of both glues on adjacent sides such that it could bind in this location creates a situation in which the assembly is not terminal if and only if the corresponding 3SAT problem has a satisfying solution. Thus, the complement of terminality-from-assembly is NP-hard.

**Theorem 4** is proven by Lemmas 3 and 4.

### Acknowledgements

Funding was provided by National Science Foundation (US) (Grant Nos. CCF-1422152, CAREER-1553166).

### References

Aichholzer O, Biro M, Demaine ED, Demaine ML, Eppstein D, Fekete SP, Hesterberg A, Kostitisyn I, Schmidt C (2017) Folding polyminoes into (poly) cubes. arXiv preprint arXiv:1712.09317

Aloupis G, Benbernou N, Bose PK, Collette S, Demaine ED, Demaine M, Douieb K, Dujmovic V, Iacono J, Langerman S et al (2011) Common unfoldings of polyminoes and polycubes. In: Akiyama J, Bo J, Kano M, Tan X (eds) Computational geometry, graphs and applications. Springer, Berlin, pp 44–54 (2011)

Barish RD, Schulman R, Rothemund PWK, Winfree E (2009) An information-bearing seed for nucleating algorithmic self-assembly. Proc Natl Acad Sci 106(15):6054–6059

Crescenzi P, Goldman D, Papadimitriou C, Piccolboni A, Yannakakis M (1998) On the complexity of protein folding. J Comput Biol 5(3):423–465

Fochtman T, Hendrickx J, Padilla JE, Patitz MJ, Rogers TA (2015) Signal transmission across tile assemblies: 3D static tiles simulate active self-assembly by 2D signal-passing tiles. Nat Comput 14(2):251–264

Fraenkel AS (1993) Complexity of protein folding. Bull Math Biol 55(6):1199–1210

Hendrickx J, Patitz MJ, Rogers TA (2017) Reflections on tiles (in self-assembly). Nat Comput 16(2):295–316

Jonoska N, Karpenko D (2014a) Active tile self-assembly, part 1: universality at temperature 1. Int J Found Comput Sci 25(02):141–163
Jonoska N, Karpenko D (2014b) Active tile self-assembly, part 2: self-similar structures and structural recursion. Int J Found Comput Sci 25(02):165–194
Jonoska N, McColm GL (2005) A computational model for self-assembling flexible tiles. In: Proceedings of the 4th international conference on unconventional computation. Springer, Berlin, pp 142–156 (2005)
Jonoska N, McColm GL (2009) Complexity classes for self-assembling flexible tiles. Theor Comput Sci 410(4–5):332–346
Liu W, Zhong H, Wang R, Seeman NC (2011) Crystalline two-dimensional DNA-origami arrays. Angew Chem Int Ed 50(1):264–267
Padilla JE, Patitz MJ, Pena R, Schweller RT, Seeman NC, Sheline R, Summers SM, Zhong X (2013) Asynchronous signal passing for tile self-assembly: fuel efficient computation and efficient assembly of shapes. In: Mauri G, Dennunzio A, Manzoni L, Porreca AE (eds) UCNC. Lecture notes in computer science. Springer, Berlin, vol 7956, pp 174–185 (2013)
Padilla JE, Patitz MJ, Schweller RT, Seeman NC, Summers SM, Zhong X (2014) Asynchronous signal passing for tile self-assembly: fuel efficient computation and efficient assembly of shapes. Int J Found Comput Sci 25(4):459–488
Rothemund PWK (2005) Design of dna origami. In: ICCAD ’05: proceedings of the 2005 IEEE/ACM international conference on computer-aided design, pp 471–478. IEEE Computer Society, Washington, DC, USA (2005)
Rothemund PWK (2006) Folding DNA to create nanoscale shapes and patterns. Nature 440:297–302
Winfree E (1998) Algorithmic self-assembly of DNA. Ph.D. thesis, California Institute of Technology

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.