Self-Assembly of Shapes at Constant Scale using Repulsive Forces[⋆]

Austin Luchsinger[*], Robert Schweller[*], and Tim Wylie[*]

[*]Department of Computer Science
University of Texas - Rio Grande Valley
Edinburg, TX 78539, USA
{austin.luchsinger01,robert.schweller,timothy.wylie}@utrgv.edu

Abstract. The algorithmic self-assembly of shapes has been considered in several models of self-assembly. For the problem of *shape construction*, we consider an extended version of the Two-Handed Tile Assembly Model (2HAM), which contains positive (attractive) and negative (repulsive) interactions. As a result, portions of an assembly can become unstable and detach. In this model, we utilize fuel-efficient computation to perform Turing machine simulations for the construction of the shape. In this paper, we show how an arbitrary shape can be constructed using an asymptotically optimal number of distinct tile types (based on the shape's Kolmogorov complexity). We achieve this at $O(1)$ scale factor in this straightforward model, whereas all previous results with sublinear scale factors utilize powerful self-assembly models containing features such as staging, tile deletion, chemical reaction networks, and tile activation/deactivation. Furthermore, the computation and construction in our result only creates constant-size garbage assemblies as a byproduct of assembling the shape.

1 Introduction

A fundamental question within the field of self-assembly, and perhaps the most fundamental, is how to efficiently self-assemble general shapes with the smallest possible set of system monomers. This question has been considered in multiple models of self-assembly. Soloveichek and Winfree [16] first showed that any shape S, if scaled up sufficiently, is self-assembled within the *abstract tile assembly model* (aTAM) using $O(\frac{K(S)}{\log K(S)})$ tile types, where $K(S)$ denotes the *Kolmogorov* or *descriptional* complexity of shape S with respect to some universal Turing machine, which matches the lower bound for this problem. This seminal result presented a concrete connection between the descriptional complexity of a shape and the efficiency of self-assembling the shape, and represents an elegant example of the potential connections between algorithmic processes and the self-assembly of matter. The only drawback with this result is the extremely large scale factor

[⋆] This research was supported in part by the National Science Foundation Grant CCF-1555626.

required by construction: the scale factor to build a shape S is at least linear in $|S|$, and is typically far greater in their construction. To lay claim as a true universal shape building scheme for potential experimental application, a much smaller scale factor is needed. Unfortunately, example shapes exist (long thin rectangles for example) which prove that the aTAM cannot build all shapes at $o(|S|)$ scale in the minimum possible $O(\frac{K(S)}{\log K(S)})$ tile complexity. This motivates the quest for small scale factors in more powerful self-assembly models.

The next result by Demaine, Patitz, Schweller, and Summers [5] considers general shape assembly within the *staged RNAse* self-assembly model. In this model, system tiles are separated into separate bins and mixed over distinct stages of the algorithm in a way that models realistic laboratory operations. In addition, each tile type in this model is of type DNA or RNA, and the staging permits the addition of an RNAse enzyme at any step in the staging, thereby dissolving all tiles of type RNA, leaving DNA tiles untouched. By adding the powerful operations of separate bins, sequential stages, and tile deletion, [5] achieves general shape construction within optimal $O(\frac{K(s)}{\log K(S)})$ tile complexity using only a constant number of bins and stages, and only a logarithmic scale factor. This leap in scale factor reduction constituted a great improvement, but required a very powerful model with both staging and tile dissolving. In addition, the holy grail of $O(1)$ scale factor remained elusive.

The next entry into the quest for Kolmogorov optimal shape assembly at small scale comes from a recent work by Schiefer and Winfree [14]. Schiefer and Winfree introduce the *chemical reaction network tile assembly model* (CRN-TAM) in which chemical reaction networks and abstract tile assembly systems combine and interact by allowing CRN species to activate and deactivate tiles, while tile attachments may introduce CRN species. This powerful interaction allowed the construction of Kolmogorov optimal systems for the assembly of general shapes at $O(1)$ scale. Although the result provides a great scale factor, the CRN-TAM constitutes a substantial jump in model complexity and power.

In this paper we study the optimal shape building problem within one of the simplest, and most well studied models of self-assembly: *the two handed tile assembly model* (2HAM), where system monomers are 4-sided tiles with glue types on each edge. Assembly in the 2HAM proceeds whenever two previously assembled conglomerations of tiles, or assemblies, collide along matching glue types whose strength sums to some temperature threshold. Our only addition to the model is the allowance of negative strength (i.e., *repulsive*) glues, an admittedly powerful addition based on recent work [6, 9–11, 15], but an addition motivated by biology [12] that maintains the *passive* nature of the model as system monomers are static, state-less pieces that simply attract or repulse based solely on surface chemistry (Figure 1). While the negative glue 2HAM has been used for works such as fuel-efficient computation [15] and recently universal shape replication [1], it is also one of the simplest models where the general shape assembly problem has been considered. Our result is on par with the best possible result: we show that any connected shape S is self-assembled at $O(1)$-

scale in the negative glue 2HAM within $O(\frac{K(S)}{\log K(S)})$ tile types, which is met by a matching lower bound.

Our Approach. We achieve our result by combining the *fuel efficient Turing machine* construction published in SODA 2013, [15], with a number of novel negative glue based gadgets. At a high level, the fuel efficient Turing machine system extracts a description of a path that walks the pixels of the constant-scaled shape from a compressed initial binary string. From there, the steps of the path are translated into *walker* gadgets which conceptually walk along the surface of the growing path and eventually deposit an additional pixel in the specified direction, with the aid of *path extension* gadgets. When all pixels have been placed, the path through the shape is filled, resulting in a scaled version of the original shape.

Additional Related Work. Additional work has considered assembly of $O(1)$-scaled shapes by breaking the assembly process up into a number of distinct stages. In particular, [3] introduce the *staged self-assembly* model in which intermediate tile assemblies grow in separate bins and are mixed and split over a sequence of distinct stages. This approach is applied to achieve $O(1)$-scaled shapes with $O(1)$ tiles types, but a large number of bins and stages which encode the target shape. In [4] this approach is pushed further to achieve tradeoffs in terms of bin complexity and stage complexity, while maintaining construction of a final assembly with no unbonded edges. In [8] similar constant-scale results are obtained in the *step-wise self-assembly* model in which tile sets are added in sequence to a growing seed assembly. Finally, in [17] $O(1)$-scaled shapes are assembled with $O(1)$ tile types by simply adjusting the temperature of a given system over multiple assembly stages. While each of above *staged* approaches offers important algorithmic insights, the large number of stages required by each makes the approaches infeasible for large shapes. Furthermore, the system complexity of these systems (which includes the staging algorithms) greatly exceeds the descriptional complexity of the goal shape in a typical case.

Paper layout. Our construction consists of a number of detailed gadgets for specific tasks. Presentation is thus organized incrementally to walk through a version of each gadget (with symmetry there may be multiple). Section 2 gives the preliminary definitions and background. In Section 3 we provide a high-level overview of the entire process as a guide for the rest of the paper. Some of the details of our construction are shown in Section 4 with the construction gadgets and how to construct a line of the path. Section 5 provides the analysis of our construction, with the lower bound on tile complexity for shape assembly presented in Section 6, and details for pushing our construction to achieve a matching upper bound in Section 7. Then we conclude in Section 8.

2 Definitions and Model

In this section we first define the two-handed tile self-assembly model with both negative and positive strength glue types. We also formulate the problem of

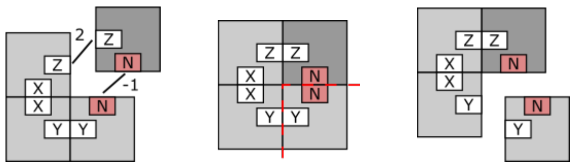


Fig. 1: This figure introduces notation for our constructions, as well as a simple example of negative glues. On each tile, the glue label is presented. Red (shaded) labels represent negative glues, and the relevant glue strengths for the tiles can be found in the captions. For caption brevity, for a glue type X we denote $str(X)$ simply as X (e.g., $X + Y = str(X) + str(Y)$). In this temperature $\tau = 1$ example, $X = 2$, $Y = 1$, $Z = 2$ and $N = -1$. (a) The three tile assembly on the left attaches with the single tile with strength $Z + N = 2 - 1 = \tau$ resulting in the 2×2 assembly shown in (b). However, this 2×2 assembly is unstable along the cut shown by the dotted line, since $Y + N = 1 - 1 < \tau$. Then the assembly is breakable into the assemblies shown in (c).

designing a tile assembly system that constructs a constant-scaled shape given the optimal description of that shape.

Tiles and Assemblies. A tile is an axis-aligned unit square centered at a point in $\mathbb{Z}^2$, where each edge is labeled by a *glue* selected from a glue set Π. A *strength function* $\mathrm{str} : \Pi \to \mathbb{N}$ denotes the *strength* of each glue. Two tiles equal up to translation have the same *type*. A *positioned shape* is any subset of $\mathbb{Z}^2$. A *positioned assembly* is a set of tiles at unique coordinates in $\mathbb{Z}^2$, and the positioned shape of a positioned assembly A is the set of those coordinates.

For a given positioned assembly Υ, define the *bond graph* G_Υ to be the weighted grid graph in which each element of Υ is a vertex and the weight of an edge between tiles is the strength of the matching coincident glues or 0.[1] A positioned assembly C is said to be *τ-stable* for positive integer τ provided the bond graph G_C has min-cut at least τ, and C is said to be *connected* if every pair of vertices in G_C has a connecting path using only positive strength edges.

For a positioned assembly A and integer vector $\boldsymbol{v} = (v_1, v_2)$, let $A_{\boldsymbol{v}}$ denote the positioned assembly obtained by translating each tile in A by vector $\boldsymbol{v}$. An *assembly* is a translation-free version of a positioned assembly, formally defined to be a set of all translations $A_{\boldsymbol{v}}$ of a positioned assembly A. An assembly is τ-stable if and only if its positioned elements are τ-stable. An assembly is *connected* if its positioned elements are connected. A *shape* is the set of all integer translations for some subset of $\mathbb{Z}^2$, and the shape of an assembly A is defined to be the set of the positioned shapes of all positioned assemblies in A. The *size* of either an assembly or shape X, denoted as $|X|$, refers to the number of elements of any positioned element of X.

[1] Note that only matching glues of the same type contribute a non-zero weight, whereas non-equal glues always contribute zero weight to the bond graph. Relaxing this restriction has been considered as well [2].

Breakable Assemblies. An assembly is τ-*breakable* if it can be cut into two pieces along a cut whose strength sums to less than τ. Formally, an assembly C is *breakable* into assemblies A and B if A and B are connected, and the bond graph $G_{C'}$ for some assembly $C' \in C$ has a cut (A', B') for $A' \in A$ and $B' \in B$ of strength less than τ. We call A and B *pieces* of the breakable assembly C.

Combinable Assemblies. Two assemblies are τ-*combinable* provided they may attach along a border whose strength sums to at least τ. Formally, two assemblies A and B are τ-*combinable* into an assembly C provided $G_{C'}$ for any $C' \in C$ has a cut (A', B') of strength at least τ for some $A' \in A$ and $B' \in B$. We call C a *combination* of A and B.

Note that A and B may be combinable into an assembly that is not stable (and thus breakable). This is a key property that is leveraged throughout our constructions. See Figure 1 for an example. For a system $\Gamma = (T, \tau)$, we say $A \rightarrow_1^{\Gamma} B$ for assemblies A and B if either A is τ-breakable into pieces that include B, or A is τ-combinable with some producible assembly to yield B, or if $A = B$. Intuitively this means that A may grow into assembly B through one or fewer combination or break reactions. We define the relation $\rightarrow^{\Gamma}$ to be the transitive closure of $\rightarrow_1^{\Gamma}$, ie., $A \rightarrow^{\Gamma} B$ means that A may grow into B through a sequence of combination or break reactions.

Producibility and Unique Assembly. A *two-handed tile assembly system (2HAM system)* is an ordered pair (T, τ) where T is a set of single tile assemblies, called the *tile set*, and $\tau \in \mathbb{N}$ is the *temperature*. Assembly proceeds by repeated combination of assembly pairs, or breakage of unstable assemblies, to form new assemblies starting from the initial tile set. The *producible assemblies* are those constructed in this way. Formally:

Definition 1 (2HAM Producibility). *For a given 2HAM system $\Gamma = (T, \tau)$, the set of* producible assemblies *of Γ, denoted $\texttt{PROD}_{\Gamma}$, is defined recursively:*

- *(Base) $T \subseteq \texttt{PROD}_{\Gamma}$*
- *(Combinations) For any $A, B \in \texttt{PROD}_{\Gamma}$ such that A and B are τ-combinable into C, then $C \in \texttt{PROD}_{\Gamma}$.*
- *(Breaks) For any assembly $C \in \texttt{PROD}_{\Gamma}$ that is τ-breakable into A and B, then $A, B \in \texttt{PROD}_{\Gamma}$.*

Definition 2 (Terminal Assemblies). *A* terminal *assembly of a 2HAM system is a producible assembly that cannot break and cannot combine with any other producible assembly. Formally, an assembly $A \in \texttt{PROD}_{\Gamma}$ of a 2HAM system $\Gamma = (T, \tau)$ is* terminal *provided A is τ-stable (will not break) and not τ-combinable with any producible assembly of Γ (will not combine).*

Definition 3 (Unique Assembly - with bounded garbage). *A 2HAM system* uniquely *produces an assembly A if all producible assemblies have a forward growth path towards the terminal assembly A, with the possible exception of some $O(1)$-sized producible assemblies. Formally, a 2HAM system $\Gamma = (T, \tau)$* uniquely *produces an assembly A provided that A is terminal, and for some constant c for all $B \in \texttt{PROD}_{\Gamma}$ such that $|B| \geq c$, $B \rightarrow^{\Gamma} A$.*

Definition 4 (Unique Shape Assembly - with bounded garbage). *A 2HAM system uniquely produces a shape S if all producible assemblies have a forward growth path to a terminal assembly of shape S with the possible exception of some $O(1)$-sized producible assemblies. Formally, a 2HAM system $\Gamma = (T, \tau)$* uniquely assembles *a finite shape S if for some constant c for every $A \in \texttt{PROD}_{\Gamma}$ such that $|A| \geq c$, there exists a terminal $A' \in \texttt{PROD}_{\Gamma}$ of shape S such that $A \rightarrow^{\Gamma} A'$.*

Definition 5 (Kolmogorov Complexity). *The* Kolmogorov complexity *(or* descriptional complexity*) of a shape S with respect to some fixed universal Turing machine U is the smallest bit string such that U outputs a list of exactly the positions in some translation of shape S when provided the bit string as input. We denote this value as $K(S)$.*

3 Concept/Construction Overview

This section presents a high-level overview of the shape construction process. First, we will present the conceptual overview, which explains the fundamental ideas behind our shape self-assembly process. Then, we will show a high-level look at how our construction implements this process.

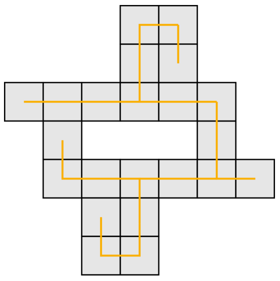
3.1 Conceptual Overview

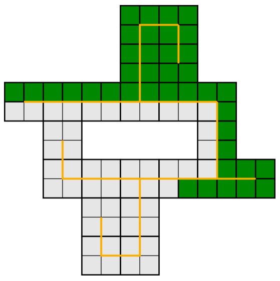
Starting with the Kolmogorov-optimal description of a shape (as a base b string, $b > 2$), we simulate a Turing machine which converts any base b string into its equivalent base 2 representation (Sec. 7) We then simulate another Turing machine that takes the binary description of a shape, finds a spanning tree for that shape, and outputs a path around that spanning tree as a set of instructions (forward, left, right) starting from a beginning node on the perimeter.

A simple depth-first search will find the spanning tree for any shape. Scaling the shape to scale 2 creates a perimeter *path* that outlines the spanning tree, and assembles the shape. Scaling again, this time by a multiple of 3, now allows space for the perimeter path with an equal-sized space buffer on both sides (Fig. 2). This buffer is required as it allows sufficient space for our construction gadgets to "walk" along the perimeter path being built.

Process Overview:

1. Given the Kolmogorov-optimal description of a shape, run a base conversion Turing machine to get its binary equivalent.
2. Given that binary string, run another Turing machine that outputs the description of a path around the shape's spanning tree as a set of instructions (forward, left, right).
3. Given those instructions, build the path. Our construction begins with a *tape* containing this *path* description for a scale 24 shape.


(a) Non-scaled shape X with spanning tree.


(b) Shape X at scale 2 with spanning tree.

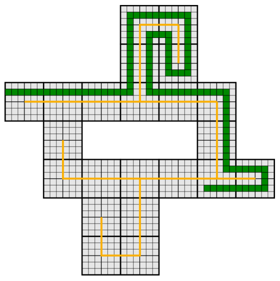

(c) Shape X at scale 6 with spanning tree.

Fig. 2: The Turing machine calculates a spanning tree of the tiles in the shape (a), scales the shape in order to allow a path around the spanning tree (b), and further scales the shape for the gadgets (c).

3.2 Construction Overview

The construction overview begins at step 3 of the conceptual overview, using the output from step 2. Throughout this paper, we will be referring to this output as the *tape*, meaning the fuel-efficient Turing machine tape with *path*-building instructions encoded on it. This *tape* is detailed in Section 4.

Construction Steps Overview:

1. **Overlay.** The overlay process is the first step in shape construction. Figure 3a-c shows an abstraction of how the output from step 2 in the concept overview gets covered during the overlay process. The overlay initiator gadget can only attach to a completed tape. This begins a series of cooperative attachments that will cover the tape. Each bit of information on the tape is covered by its corresponding overlay piece, and thus is readable on the top of the overlay. The overlay process is finished once the entire tape is covered.
2. **Reading.** After the overlay process is complete, information can be extracted from the tape through the read process (Figs. 3d-f). Information can only be extracted from the covered leftmost section of the tape if it has not already been read. When a tape section is read, information is extracted from the tape and a corresponding information block is created.
3. **Information Walking.** Once the information block is created, it begins walking until it reaches the end of the tape/path (Figs. 3g-i). Walking gadgets allow the information to travel down the entire path.
4. **Path Extension.** When an information block cannot travel any further, the path is extended (Figs. 3j-l). The path can be extended forward, left, or right. The direction of the path extension is dependent on which information block is at the end of the path. After the path is extended, the information block is removed from the path.
5. **Tape Reduction.** Once information is extracted from the tape and sent down the path, one tape section is removed (Figs. 3j-l). Only tape sections

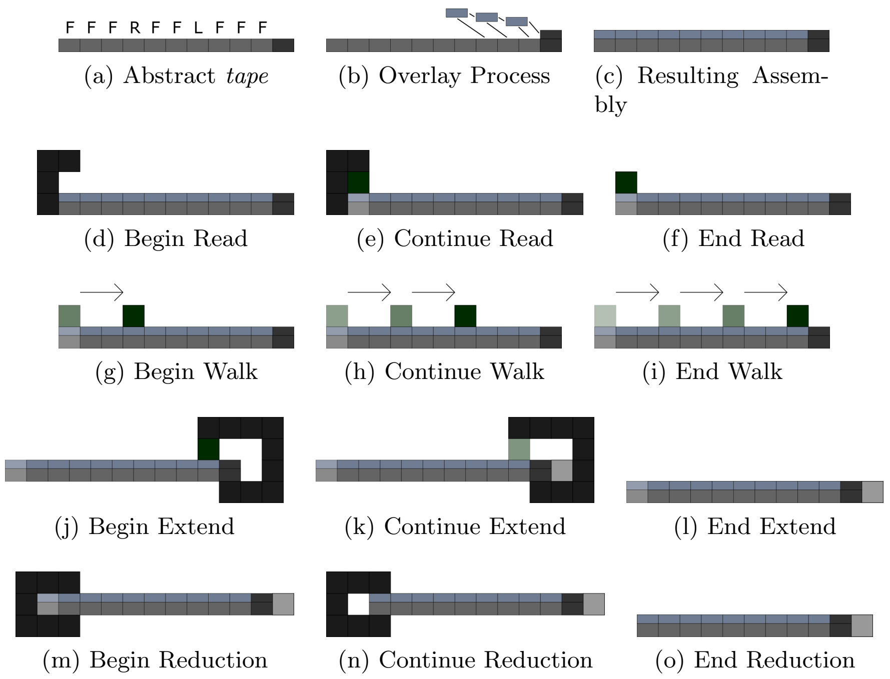


Fig. 3: (a)-(c) The *overlay* process covers the tape while making the data readable on top. (d)-(f) *Reading* the leftmost piece of data and creating an information block (depicted in green). (g)-(i) *Information Walking* on the path to the end where the information is used. (j)-(l) When the information block reaches the end of the path, the block triggers a *Path Extension.* (m)-(o) Once the information has been read, *Tape Reduction* removes that piece of the tape.

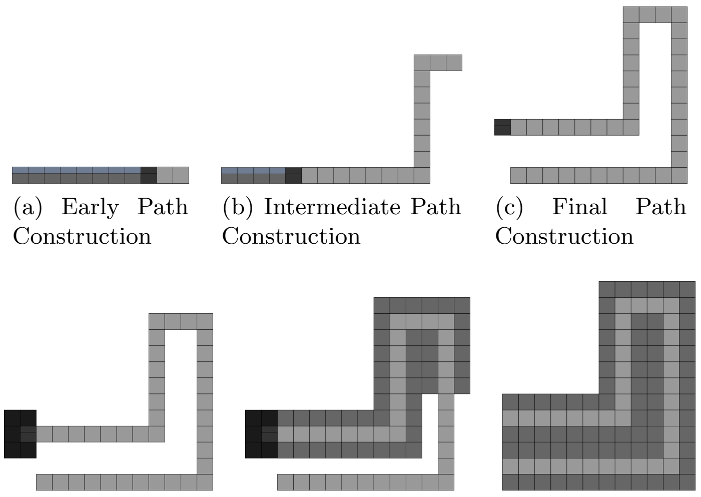
that have been read are removed, which then allows the next section to be read. This process continues until every section of the tape is read/removed.

6. **Repeat.** Repeat the tape read, information walk, path extend, and tape reduction processes until all path instructions have been read (Figs. 4a-c).
7. **Path Filling.** The final tape section that gets read begins the shape fill process (Figs. 4d-f). In this process, the path is padded with tiles which fill it in and results in the final shape.

4 Construction Details

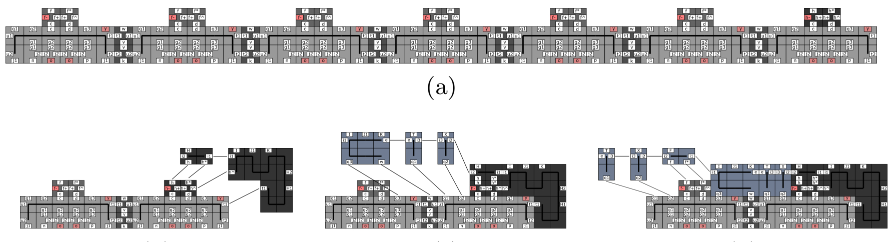
In this section, we detail the steps presented in the construction overview (Sec. 3.2). This is the process by which information is read from the *tape* and portions of the *path* are assembled.

We will also cover the gadgets required for each step, and review the tape construction from the fuel-efficient Turing machine used in [15]. This construction uses pre-constructed assemblies called gadgets. These gadgets are designed


(a) Early Path Construction (b) Intermediate Path Construction (c) Final Path Construction

(d) Begin Fill (e) Continue Fill (f) End Fill

Fig. 4: (a)-(c) The process is repeated until all information has been read/removed from the tape. (d)-(f) The final step is *Path Filling* the shape.


(a)

(b) (c) (d)

Fig. 5: (a) A completed *tape* consisting of all *forward* instructions. (b) Overlay Initiator gadget attaching to tape. (c-d) Overlay fillers begin covering all *tape* sections from right to left.

to work in a temperature $\tau = 10$ system. In our figures, a perpendicular black line through the middle of the edge of two adjacent tiles indicates a unique $2\tau = 20$ strength bond[2]. Each gadget provides a different function to the shape creation process.

Turing Machine Tape. A detailed look at a fuel-efficient Turing machine *tape* is seen in Figure 5a. Notice each tape section has a pair of tiles on top of it where the information is stored. When talking about the *tape* from Section 3.2, each pair of dark grey tiles on top of the tape sections represents a piece of information describing the *path*.

[2] The strongest detaching force used in our construction is a τ strength detachment, and since the internal bonds of our gadgets are meant to withstand even the strongest repulsive force, it follows that those bonds must be of strength at least 2τ.

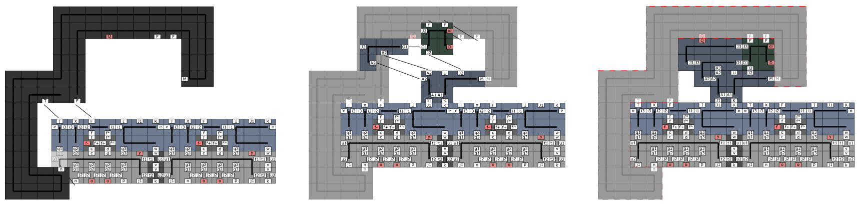

(a) (b) (c)

Fig. 6: (a) The Read Gadget attaches ($n + T + F = 2 + 7 + 1 \geq \tau$). In (b) the first form of an information block attaches ($F + F + J2 = 1 + 1 + 8 \geq \tau$). Since the *forward* version of the read gadget was used, the *forward* information block is placed. After the information block is placed, the penultimate read-helper attaches ($A2 + A2 + O1 = 2 + 2 + 7 \geq \tau$). (c) After all read helpers have attached, the read gadget becomes unstable ($F + F + M + n + T + F + Q = 1 + 1 + 1 + 2 + 7 + 1 - 7 \leq \tau$).

The Overlay Initiator Gadget attaches to the end of the completed *tape*, and begins the overlay process (Fig. 5b-d). Each bit of information on the tape is covered by a corresponding overlay section, allowing the information to be read on top of the overlay. This process continues, section by section, until the entire tape is covered. Once finished, the overlay layer will act as an interface, allowing the gadgets to use the information on the *tape*.

Read. The read gadget is required for "reading" the Turing machine *tape*. Essentially, this gadget extracts the information that is relayed from the *tape* through the overlay blocks. The read process (Fig. 6a-c) can only begin if the leftmost tape section has not previously been read. Once attached, the gadget allows the attachment of an information block (corresponding to the information being read) that will be used to carry the build instructions through the rest of our construction. Once the information block is present, the remaining read-helpers can attach. The final helper destabilizes the read gadget, allowing it to fall off and expose the newly attached information block. The read gadget was designed to produce this information block, alter the *tape* section that is being read (making it unreadable), and then detach from the assembly. This design ensures that each *tape* section is only read once, and allows us to transfer the instructions to other locations in our construction via the walking gadgets.

Information Walking. The walking gadgets begin the information walking process (Fig. 7), which allows instructions to travel throughout our construction. After a tape section has been read and an information block has been placed, a walking gadget can attach. Once attached, the walking gadget allows a new information block (of the same type) to attach, while also detaching the the previous information block. Notice that this detachment will always be $O(1)$ size. After the previous information is removed, the walking gadget detaches as well, allowing the new info block to interact with other gadgets. Thus, the same information has traveled from the *tape*, through the overlay, and is now traveling along the *tape*. This process is repeated until the information has traveled to the

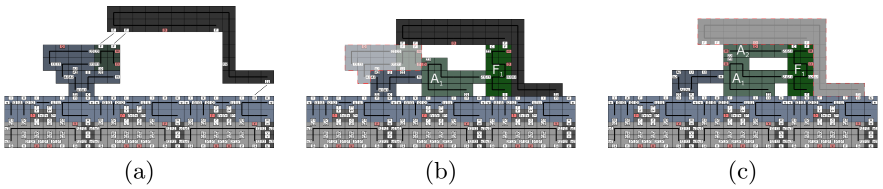


Fig. 7: (a) A Walking Gadget (specific to the information block) attaches to the overlay and the information block ($F+F+J1 = 1+1+8 \geq \tau$). (b) The negative interaction between the D glues destabilizes the old information block, along with the two walking-helpers ($J2+A2+A2+F+F+D = 8+2+2+1+1-7 \leq \tau$). Notice that two helpers remain attached to the tape, as they will be used later in the construction. (c) Once the second walking-helper is attached, the walking gadget becomes unstable ($F+O2+J1+D = 1+7+8-7 \leq \tau$).

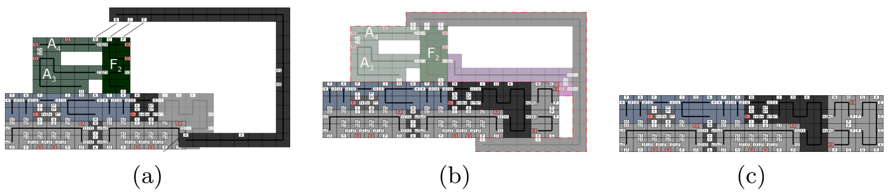
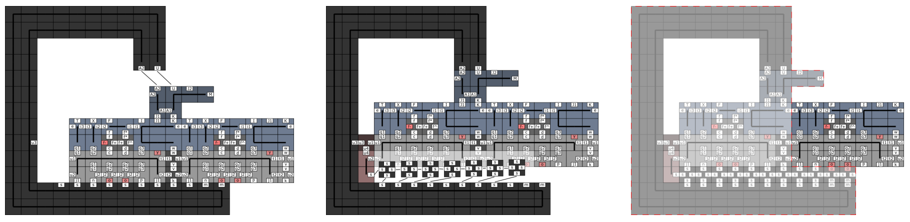


Fig. 8: (a) The forward-extension gadget attaches to the information block and Turing tape ($B+C+F+p = 3+4+1+2 \geq \tau$). (b) The second extension-helper comes with the negative D glue that causes targeted destabilization ($X+p+J1+X+D = 2+2+8+2-7 \leq \tau$). The extension gadget and its helpers, along with the information block and its helpers are no longer stable along their tape-overlay edges. (c) The final result is a one path-pixel extension of the path.

end of the *path*, at which point it is used to construct the next *path* portion. This method is desirable because it does not allow duplicate readable instructions to be attached to the path at any time.

Path Extension. After the information block has reached the end of the *path*, a path extension gadget can attach to the assembly. Once attached, the gadget allows the *path* extension process (Fig. 8) to begin, which extends the *path* in a given direction (forward, left, or right) based on the instruction carried by the information block. The extension gadget "reads" the information block, and then extends the path in the given direction. Afterwards, the extension helpers destabilize the information block and extension gadget, causing a $O(1)$ sized detachment. We designed the extension gadget to essentially replace an instruction block with a corresponding *path* portion. This design allows us to attach a $O(1)$ sized *path* portion for each instruction read from the *tape*.

Tape Reduction. After a tape section has been read, we no longer need it. Instead of continuing to grow the assembly, we can remove $O(1)$ size portions of the *tape* as it is being read. This is where the tape reduction gadget initiates the



(a) (b) (c)

Fig. 9: (a) The tape reduction gadget attaches to the read-helpers ($A2 + U = 2 + 8 \geq \tau$). (b) Filler tiles attach ($s + s = 8 + 8 \geq \tau$), and create a strong bond to the tape reduction gadget. (c) The two negative o glues cause a strong targeted destabilization of the previously read tape section ($e+u1+u2+o+o = 3+8+8-5-5 \leq \tau$).

tape reduction process (Fig. 9) mentioned in Section 3.2. The attachments left behind by the read/walk processes allow the tape reduction gadget to attach to a tape section that has already been read. The gadget then removes itself, along with the previously read tape section, exposing the next section of the tape for reading. This technique is desirable because it allows us to break apart the *tape* into $O(1)$ sized pieces as we use it. As the *tape* is reduced, the *path* continues to grow until there are no more *tape* sections to be read.

Due to page constraints, some of the construction details have been omitted (such as turning and filling). For complete details, please see the arXiv version of this paper [7].

5 Constant Scaled Shapes

In this section, we formally state the results based on our construction.

Theorem 1. *For any finite connected shape S, there exists a 2HAM system $\Gamma = (T_S, 10)$ that uniquely produces S (with $O(1)$ size bounded garbage) at a $O(1)$ scale factor, and $|T_S| = O(\frac{K(S)}{\log K(S)})$.*

Proof. We show this by constructing a 2HAM system $\Gamma = (T_S, 10)$. One portion of T_S consists of the tile types which assemble a higher base Kolmogorov-optimal description of S (Section 7). This portion of T_S consists of $O(\frac{K(S)}{\log K(S)})$ tile types, as analyzed in Section 7. Another portion of T_S consists of the tile types needed to assemble a fuel-efficient Turing machine, as described by [15], that performs a simple base conversion to binary using $O(\frac{K(S)}{\log K(S)})$ tile types, as analyzed in Section 7. The next portion of T_S consists of the tile types required to assemble another fuel-efficient Turing machine that finds and outputs the description of a path around the spanning tree of S. This Turing machine is of $O(1)$ size, and thus adds $O(1)$ tile types using the method from [15]. The final portion of T_S consists of the tile types that construct the gadgets and assemblies shown in Section 4. With the number of tile types used for computing the *path* description and for our construction process being $O(1)$, our final tile complexity is $O(\frac{K(S)}{\log K(S)})$.

Now, consider assembly A to be the fully constructed *tape* assembly (Section 4) encoded with *path*-building instructions specific to S. Also, suppose assembly B is some *terminal* assembly that has shape S at a constant scale factor.

Note that Γ follows the process detailed in Section 4. This system was designed so that two assemblies are *combinable* only if at least one of those assemblies is at most a constant size (70 tiles), and every *breakable* assembly can only break into two subassemblies if one of those assemblies is at most another constant size (118 tiles). In our construction, the only non-constant size assemblies are A, B, or some intermediate assembly that consists of some portion of the *tape*, and some partially assembled section of the final shape. Of these, B is the only terminal assembly.

While A and the intermediate assemblies continue engaging in a series of attachments and detachments, the *tape* continues to get smaller and the *path* continues to grow. The attachment and detachment of $O(1)$ size pieces with these assemblies will continue until we reach the terminal assembly B, at which time A will have been disassembled into smaller constant garbage. Therefore, we see that $A \rightarrow^{\Gamma} B$. □

6 Lower Bound

Here we present a brief argument for the lower bound of $\Omega(\frac{K(S)}{\log K(S)})$ on the tile types needed to assemble a scaling of a shape S. This argument is essentially the same as what is presented in [2, 13, 16], and we refer the reader there for a more detailed explanation.

Theorem 2. *The tile complexity in the 2HAM for self-assembling a scale-c version of a shape S at constant temperature and constant garbage is $\Omega(\frac{K(S)}{\log K(S)})$.*

Proof. Note that a 2HAM system $\Gamma = (T, \tau = O(1))$ can be uniquely represented with a string of $O(|T| \log |T|)$ bits. In particular, each tile may be encoded as a list of its 4 glues, and each glue may be represented by a $O(\log |T|)$-bit string taken from an indexing of the maximum possible $4|T|$ distinct glue types of the system. The constant bounded temperature incurs an additional additive constant. Given this representation, consider a 2HAM simulation program that inputs a 2HAM system, and outputs the positions of any uniquely produced scale-c shape (with up to $O(1)$ garbage), if one exists. This simulator, along with the $O(|T| \log |T|)$ bit encoding of a system Γ which assembles S at scale c, constitute a program which outputs the positions of S, and is thus lower bounded in bits by $K(S)$. Therefore $K(S) \leq d|T| \log |T|$ for some constant d, implying $|T| = \Omega(\frac{K(S)}{\log K(S)})$. □

7 Extension to $\frac{K(S)}{\log K(S)}$

The starting assembly for our shape construction algorithm is the *tape* assembly from [15] with a binary string as its value. For a binary string $A = a_0 \ldots a_{k-1}$, such an assembly can be constructed in a straightforward manner using $O(k)$

tile types (simply place a distinct tile for each position in the assembly, for example). However, by using a base conversion trick, we can take advantage of the fact that each tile type is asymptotically capable of representing slightly more than 1 bit in order to build the string in $O(k/\log k)$ tile types. To achieve this, first we consider the base-b representation $B = b_0 \dots b_{d-1}$ of the string A for some higher base $b > 2$. Note that the number of digits of this string is $d \leq \lceil \frac{k}{\lfloor \log_2 b \rfloor} \rceil = O(\frac{k}{\log b})$. We are able to assemble this shorter string (by brute force with distinct tile types at each position) with only $O(d)$ tile types.

Next, we consider a Turing machine which converts any base b string into its equivalent base 2 representation. Such a Turing machine can be constructed using $O(b)$ transition rules. Therefore, we can apply the result of [15] to run this Turing machine on the initial tape assembly representing string B to obtain string A. The cost of this construction in total is $O(d)$ tiles to construct the initial tape assembly, plus $O(b)$ tiles to implement the rules of the conversion Turing machine[3], for a total of $O(d + b)$ tiles.

Finally, we select $b = \lceil \frac{k}{\log k} \rceil = O(\frac{k}{\log k})$, which yields $d = O(\frac{k}{\log k - \log \log k}) = O(\frac{k}{\log k})$, implying that the entire tile cost of setting up the initial tape assembly representing binary string B is $O(b + d) = O(\frac{k}{\log k})$ tile types. In our case $k = O(K(S))$ where $K(S)$ denotes the Kolmogorov complexity of shape S for some given universal Turing machine, and so we achieve our final tile complexity of $O(\frac{K(S)}{\log K(S)})$.

8 Conclusion

In this work, we considered the optimal shape building problem in the negative glue 2-handed assembly model, and provided a system that allows the self-assembly of general shapes at scale 24. Shape construction has been studied in more powerful self-assembly models such as the staged RNA assembly model and the chemical reaction network-controlled tile assembly model. However, our result constitutes the first example of optimal general shape construction at constant scale in a *passive* model of self-assembly where no outside experimenter intervention is required, and system monomers are state-less, static pieces which interact solely based on the attraction and repulsion of surface chemistry.

Our work opens up a number of directions for future work. We have not considered a runtime model for this construction, so analyzing and improving the *running time* for constant-scaled shape self-assembly in the 2-handed assembly is one open direction. Another is determining the lowest necessary temperature and glue strengths needed for $O(1)$ scale shape construction. We use temperature value 10 for the sake of clarity, and have not attempted to optimize this value.

[3] The formal theorem statement of [15] cites the product of the states and symbols of the Turing machine as the tile type cost. However, the actual cost is the number of transition rules, which is upper bounded by this product.

References

1. Chalk, C., Demiane, E.D., Demaine, M.L., Martinez, E., Schweller, R., Vega, L., Wylie, T.: Universal shape replicators via self-assembly with attractive and repulsive forces. In: Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17) (2017)
2. Cheng, Q., Aggarwal, G., Goldwasser, M.H., Kao, M.Y., Schweller, R.T., de Espanés, P.M.: Complexities for generalized models of self-assembly. SIAM Journal on Computing 34, 1493–1515 (2005)
3. Demaine, E.D., Demaine, M.L., Fekete, S.P., Ishaque, M., Rafalin, E., Schweller, R.T., Souvaine, D.L.: Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. Natural Computing 7(3), 347–370 (2008)
4. Demaine, E.D., Fekete, S.P., Scheffer, C., Schmidt, A.: New geometric algorithms for fully connected staged self-assembly. In: DNA Computing and Molecular Programming, Lecture Notes in Computer Science, vol. 9211, pp. 104–116 (2015)
5. Demaine, E.D., Patitz, M.J., Schweller, R.T., Summers, S.M.: Self-assembly of arbitrary shapes using rnase enzymes: Meeting the kolmogorov bound with small scale factor (extended abstract). In: Proc. of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS'11) (2011)
6. Doty, D., Kari, L., Masson, B.: Negative interactions in irreversible self-assembly. Algorithmica 66(1), 153–172 (2013)
7. Luchsinger, A., Schweller, R., Wylie, T.: Self-assembly of shapes at constant scale using repulsive forces (2016), arXiv1608.04791
8. Mauch, J., Stacho, L., Stoll, C.: Step-wise tile assembly with a constant number of tile types. Natural Computing 11(3), 535–550 (2012)
9. Patitz, M.J., Rogers, T.A., Schweller, R., Summers, S.M., Winslow, A.: Resiliency to multiple nucleation in temperature-1 self-assembly. In: DNA Computing and Molecular Programming. Springer International Publishing (2016)
10. Patitz, M.J., Schweller, R.T., Summers, S.M.: Exact shapes and turing universality at temperature 1 with a single negative glue. In: DNA Computing and Molecular Programming, LNCS, vol. 6937, pp. 175–189 (2011)
11. Reif, J.H., Sahu, S., Yin, P.: Complexity of graph self-assembly in accretive systems and self-destructible systems. Theoretical Comp. Sci. 412(17), 1592–1605 (2011)
12. Rothemund, P.W.K.: Using lateral capillary forces to compute by self-assembly. Proceedings of the National Academy of Sciences 97(3), 984–989 (2000)
13. Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares (extended abstract). In: Proc. of the 32nd ACM Sym. on Theory of Computing. pp. 459–468. STOC'00 (2000)
14. Schiefer, N., Winfree, E.: Universal Computation and Optimal Construction in the Chemical Reaction Network-Controlled Tile Assembly Model, pp. 34–54. Springer International Publishing, Cham (2015)
15. Schweller, R., Sherman, M.: Fuel efficient computation in passive self-assembly. In: SODA 2013: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1513–1525. SIAM (2013)
16. Soloveichik, D., Winfree, E.: Complexity of self-assembled shapes. SIAM Journal on Computing 36(6), 1544–1569 (2007)
17. Summers, S.M.: Reducing tile complexity for the self-assembly of scaled shapes through temperature programming. Algorithmica 63(1), 117–136 (2012)