

Tilt Assembly: Algorithms for Micro-Factories that Build Objects with Uniform External Forces*

Aaron T. Becker^{†1}, Sándor P. Fekete², Phillip Keldenich³,
Dominik Krupke⁴, Christian Rieck⁵, Christian Scheffer⁶, and
Arne Schmidt⁷

- 1 Department of Electrical and Computer Engineering, University of Houston, USA
atbecker@uh.edu
- 2 Department of Computer Science, TU Braunschweig, Germany
s.fekete@tu-bs.de
- 3 Department of Computer Science, TU Braunschweig, Germany
p.keldenich@tu-bs.de
- 4 Department of Computer Science, TU Braunschweig, Germany
d.krupke@tu-bs.de
- 5 Department of Computer Science, TU Braunschweig, Germany
c.rieck@tu-bs.de
- 6 Department of Computer Science, TU Braunschweig, Germany
c.scheffer@tu-bs.de
- 7 Department of Computer Science, TU Braunschweig, Germany
arne.schmidt@tu-bs.de

Abstract

We present algorithmic results for the parallel assembly of many micro-scale objects in two and three dimensions from tiny particles, which has been proposed in the context of programmable matter and self-assembly for building high-yield micro-factories. The underlying model has particles moving under the influence of uniform external forces until they hit an obstacle; particles can bond when being forced together with another appropriate particle.

Due to the physical and geometric constraints, not all shapes can be built in this manner; this gives rise to the TILT ASSEMBLY PROBLEM (TAP) of deciding constructibility. For simply-connected polyominoes P in 2D consisting of N unit-squares (“tiles”), we prove that TAP can be decided in $O(N \log N)$ time. For the optimization variant MAXTAP (in which the objective is to construct a subshape of maximum possible size), we show *polyAPX*-hardness: unless $P=NP$, MAXTAP cannot be approximated within a factor of $N^{\frac{1}{3}}$; for tree-shaped structures, we give an $N^{\frac{1}{2}}$ -approximation algorithm. For the efficiency of the assembly process itself, we show that any constructible shape allows *pipelined* assembly, which produces copies of P in $O(1)$ amortized time, i.e., N copies of P in $O(N)$ time steps. These considerations can be extended to three-dimensional objects: For the class of polycubes P we prove that it is NP-hard to decide whether it is possible to construct a path between two points of P ; it is also NP-hard to decide constructibility of a polycube P . Moreover, it is *expAPX*-hard to maximize a path from a given start point.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Programmable matter, micro-factories, tile assembly, tilt, approximation, hardness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.11

* A full version has been made available on arxiv.org [4], <https://arxiv.org/abs/1709.06299>.

† Work from this author was partially supported by National Science Foundation IIS-1553063 and IIS-1619278.



© Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, Christian Rieck, Christian Scheffer, and Arne Schmidt;
licensed under Creative Commons License CC-BY

28th International Symposium on Algorithms and Computation (ISAAC 2017).

Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 11; pp. 11:1–11:13

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In recent years, progress on flexible construction at micro- and nano-scale has given rise to a large set of challenges that deal with algorithmic aspects of programmable matter. Examples of cutting-edge application areas with a strong algorithmic flavor include self-assembling systems, in which chemical and biological substances such as DNA are designed to form predetermined shapes or carry out massively parallel computations; and swarm robotics, in which complex tasks are achieved through the local interactions of robots with highly limited individual capabilities, including micro- and nano-robots.

Moving individual particles to their appropriate attachment locations when assembling a shape is difficult because the small size of the particles limits the amount of onboard energy and computation. One successful approach to dealing with this challenge is to use molecular diffusion in combination with cleverly designed sets of possible connections: in *DNA tile self-assembly*, the particles are equipped with sophisticated bonds that ensure that only a predesigned shape is produced when mixing together a set of tiles, see [18]. The resulting study of algorithmic tile self-assembly has given rise to an extremely powerful framework and produced a wide range of impressive results. However, the required properties of the building material (which must be specifically designed and finely tuned for each particular shape) in combination with the construction process (which is left to chemical reactions, so it cannot be controlled or stopped until it has run its course) make DNA self-assembly unsuitable for some applications.

An alternative method for controlling the eventual position of particles is to apply a uniform external force, causing all particles to move in a given direction until they hit an obstacle or another blocked particle. As two of us (Becker and Fekete, [1]) have shown in the past, combining this approach with custom-made obstacles (instead of custom-made particles) allows complex rearrangements of particles, even in grid-like environments with axis-parallel motion. The appeal of this approach is that it shifts the design complexity from the building material (the tiles) to the machinery (the environment). As recent practical work by Manzoor et al. [15] shows, it is possible to apply this to simple “sticky” particles that can be forced to bond, see Fig. 1: the overall assembly is achieved by adding particles one at a time, attaching them to the existing sub-assembly. Moreover, pipelining this process may result in efficient rates of production, see Fig. 2 [15].

One critical issue of this approach is the requirement of getting particles to their destination without being blocked by or bonding to other particles. As Fig. 3 shows, this is not always possible, so there are some shapes that cannot be constructed by Tilt Assembly.

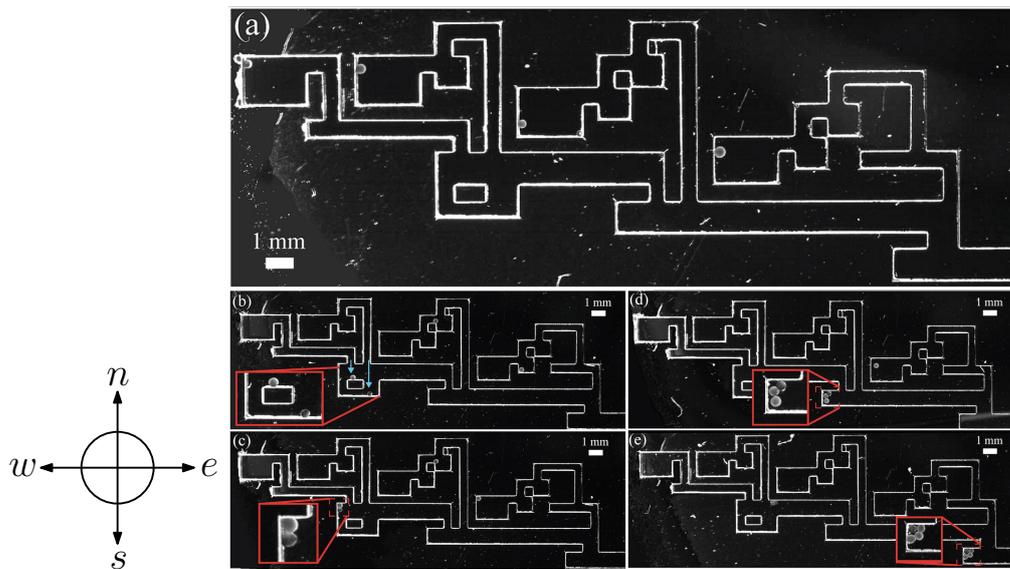
This gives rise to a variety of algorithmic questions: (1) Can we decide efficiently whether a given polyomino can be constructed by Tilt Assembly? (2) Can the resulting process be pipelined to yield low amortized building time? (3) Can we compute a maximum-size subpolyomino that can be constructed? (4) What can be said about three-dimensional versions of the problem?

1.1 Our Contribution

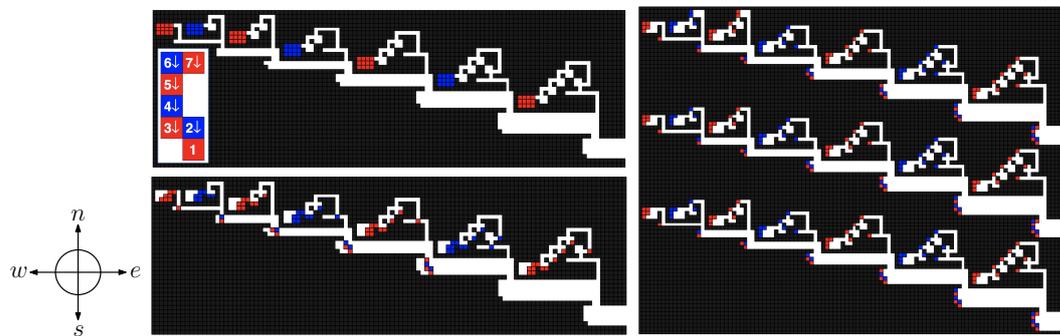
We present the results shown in Table 1.

1.2 Related Work

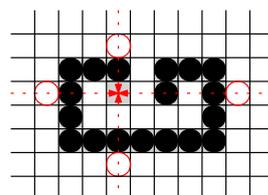
Assembling polyominoes with tiles has been considered intensively in the context of *tile self-assembly*. In 1998, Erik Winfree [18] introduced the *abstract tile self-assembly model*



■ **Figure 1** A practical demonstration of Tilt Assembly based on alginate (i.e., a gel made by combining a powder derived from seaweed with water) particles [15]. (a) Alginate particles in initial positions. (b) After control moves of $\langle e, s, w, n, e, s \rangle$ (for east, south, west, north), the alginate microrobots move to the shown positions. (c) After $\langle w, n \rangle$ inputs, the system produces the first multi-microrobot polyomino. (d) The next three microrobot polyominoes are produced after applying multiple $\langle e, s, w, n \rangle$ cycles. (e) After the alginate microrobots have moved through the microfluidic factory layout, the final 4-particle polyomino is generated.



■ **Figure 2** (Top left) Initial setup of a seven-tile polyomino assembly; the composed shape is shown enlarged on the lower left. The bipartite decomposition into blue and red particles is shown for greater clarity, but can also be used for better control of bonds. The sequence of control moves is $\langle e, s, w, n \rangle$, i.e., a clockwise order. (Bottom left) The situation after 18 control moves. (Right) The situation after 7 full cycles, i.e., after 28 control moves; shown are three parallel “factories”.



■ **Figure 3** A polyomino (black) that cannot be constructed by Tilt Assembly: the last tile cannot be attached, as it gets blocked by previously attached tiles.

■ **Table 1** Results for Tilt Assembly Problem (TAP) and its maximization variant (MAXTAP).

Dimension	Decision	Maximization	Approximation	Constructible Path
2D (simple)	$O(N \log N)$ (Sec. 3)	<i>polyAPX</i> -hard	$\Omega(N^{1/3}), O(\sqrt{N})$ (Sec. 4)	$O(N \log N)$ (Sec. 4)
3D (general)	NP-hard (Sec. 5)	<i>polyAPX</i> -hard	$\Omega(N^{1/3}), -$ (Sec. 4)	NP-hard (Sec. 5)

(aTAM), in which tiles have glue types on each of the four sides and two tiles can stick together if their glue type matches and the bonding strength is sufficient. Starting with a *seed tile*, tiles will continue to attach to the existing partial assembly until they form a desired polyomino; the process stops when no further attachments are possible. Apart from the aTAM, there are various other models like the *two-handed tile self-assembly model* (2HAM) [8] and the *hierarchical tile self-assembly model* [9], in which we have no single seed but pairs of subassemblies that can attach to each other. Furthermore, the *staged self-assembly model* [10, 11] allows greater efficiency by assembling polyominoes in multiple bins which are gradually combined with the content of other bins.

All this differs from the model in Tilt Assembly, in which each tile has the same glue type on all four sides, and tiles are added to the assembly one at a time by attaching them from the outside along a straight line. This approach of externally movable tiles has actually been considered in practice at the microscale level using biological cells and an MRI, see [12], [13], [5]. Becker et al. [6] consider this for the assembly of a magnetic *Gauß gun*, which can be used for applying strong local forces by very weak triggers, allowing applications such as micro-surgery.

Using an external force for moving the robots becomes inevitable at some scale because the energy capacity decreases faster than the energy demand. A consequence is that all non-fixed robots/particles perform the same movement, so all particles move in the same direction of the external force until they hit an obstacle or another particle. These obstacles allow shaping the particle swarm. Designing appropriate sets of obstacles and moves gives rise to a range of algorithmic problems. Deciding whether a given initial configuration of particles in a given environment can be transformed into a desired target configuration is NP-hard [1], even in a grid-like setting, whereas finding an optimal control sequence is shown to be PSPACE-complete by Becker et al. [2]. However, if it is allowed to *design* the obstacles in the first place, the problems become much more tractable [1]. Moreover, even complex computations become possible: If we allow additional particles of double size (i.e., two adjacent fields), full computational complexity is achieved, see Shad et al. [16]. Further related work includes gathering a particle swarm at a single position [14] and using swarms of very simple robots (such as Kilobots) for moving objects [7]. For the case in which human controllers have to move objects by such a swarm, Becker et al. [3] study different control options. The results are used by Shahrokhi and Becker [17] to investigate an automatic controller.

Most recent and most closely related to our paper is the work by Manzoor et al. [15], who use global control to assemble polyominoes in a pipelined fashion: after constructing the first polyomino, each cycle of a small control sequence produces another polyomino. However, the algorithmic part is purely heuristic; providing a thorough understanding of algorithms and complexity is the content of our paper.

2 Preliminaries

Polyomino. For a set $P \subset \mathbb{Z}^2$ of N grid points in the plane, the graph G_P is the induced grid graph, in which two vertices $p_1, p_2 \in P$ are connected if they are at unit distance. Any set P with connected grid graph G_P gives rise to a *polyomino* by replacing each point $p \in P$ by a unit square centered at p , which is called a *tile*; for simplicity, we also use P to denote the polyomino when the context is clear, and refer to G_P as the dual graph of the polyomino; P is *tree-shaped*, if G_P is a tree.

A polyomino is called *hole-free* or *simple* if and only if the grid graph induced by $\mathbb{Z}^2 \setminus P$ is connected.

Blocking sets. For each point $p \in \mathbb{Z}^2$ we define *blocking sets* $N_p, S_p \subseteq P$ as the set of all points $q \in P$ that are above or below p and $|p_x - q_x| \leq 1$. Analogously, we define the blocking sets $E_p, W_p \subseteq P$ as the set of all points $q \in P$ that are to the right or to the left of p and $|p_y - q_y| \leq 1$.

Construction step. A *construction step* is defined by a direction (north, east, south, west, abbreviated by n, e, s, w) from which a tile is added and a latitude/longitude l describing a column or row. The tile arrives from (l, ∞) for north, (∞, l) for east, $(l, -\infty)$ for south, and $(-\infty, l)$ for west into the corresponding direction until it reaches the first grid position that is adjacent to one occupied by an existing tile. If there is no such tile, the polyomino does not change. We note that a position p can be added to a polyomino P if and only if there is a point $q \in P$ with $\|p - q\|_1 = 1$ and one of the four blocking sets, N_p, E_p, S_p or W_p , is empty. Otherwise, if none of these sets are empty, this position is *blocked*.

Constructibility. Beginning with a seed tile at some position p , a polyomino P is *constructible* if and only if there is a sequence $\sigma = ((d_1, l_1), (d_2, l_2), \dots, (d_{N-1}, l_{N-1}))$, such that the resulting polyomino P' , induced by successively adding tiles with σ , is equal to P . We allow the constructed polyomino P' to be a translated copy of P . Reversing σ yields a *decomposition sequence*, i.e., a sequence of tiles getting removed from P .

3 Constructibility of Simple Polyominoes

In this section we focus on hole-free (i.e., simple) polyominoes. We show that the problem of deciding whether a given polyomino can be constructed can be solved in polynomial time. This decision problem can be defined as follows.

► **Definition 1** (TILT ASSEMBLY PROBLEM). Given a polyomino P , the TILT ASSEMBLY PROBLEM (TAP) asks for a sequence of tiles constructing P , if P is constructible.

3.1 A Key Lemma

A simple observation is that construction and (restricted) decomposition are the same problem. This allows us to give a more intuitive argument, as it is easier to argue that we do not lose connectivity when removing tiles than it is to prove that we do not block future tiles.

► **Theorem 2.** *A polyomino P can be constructed if and only if it can be decomposed using a sequence of tile removal steps that preserve connectivity. A construction sequence is a reversed decomposition sequence.*



(a) Removing t destroys decomposability. The polyomino can be decomposed by starting with the three tiles above t .

(b) Removing the red convex tile leaves the polyomino non-decomposable; it can be decomposed by starting from the bottom or the sides.

■ **Figure 4** Two polyominoes and their convex tiles (white). (a) Removing non-convex tiles may destroy decomposability. (b) In case of non-simple polygons we may not be able to remove convex tiles.

Proof. To prove this theorem, it suffices to consider a single step. Let P be a polyomino and t be a tile that is removed from P into some direction l , leaving a polyomino P' . Conversely, adding t to P' from direction l yields P , as there cannot be any tile that blocks t from reaching the correct position, or we would not be able to remove t from P in direction l . ◀

For hole-free polyominoes we can efficiently find a construction/decomposition sequence if one exists. The key insight is that one can greedily remove *convex* tiles. A tile t is said to be convex if and only if there is a 2×2 square solely containing t ; see Fig. 4. If a convex tile is *not* a cut tile, i.e., it is a tile whose removal does *not* disconnect the polyomino, its removal does not interfere with the decomposability of the remaining polyomino.

This conclusion is based on the observation that a minimal cut (i.e., a minimal set of vertices whose removal leaves a disconnected polyomino) of cardinality two in a hole-free polyomino always consists of two (possibly diagonally) adjacent tiles. Furthermore, we can always find such a removable convex tile in any decomposable hole-free polyomino. This allows us to devise a simple greedy algorithm.

We start by showing that if we find a non-blocked convex tile that is not a cut tile, we can simply remove it. It is important to focus on convex tiles, as the removal of non-convex tiles can harm the decomposability: see Fig. 4a for an illustration. In non-simple polyominoes, the removal of convex tiles can destroy decomposability, as demonstrated in Fig. 4b.

► **Lemma 3.** *Consider a non-blocked non-cut convex tile t in a hole-free polyomino P . The polyomino $P - t$ is decomposable if and only if P is decomposable.*

Proof. The first direction is trivial: if $P - t$ is decomposable, P is decomposable as well, because we can remove the non-blocked tile t first and afterwards use the existing decomposition sequence for $P - t$. The other direction requires some case distinctions. Suppose for contradiction that P is decomposable but $P - t$ is not, i.e., t is important for the later decomposition.

Consider a valid decomposition sequence for P and the first tile t' we cannot remove if we were to remove t in the beginning. W.l.o.g., let t' be the first tile in this sequence (removing all previous tiles obviously does not destroy the decomposability). When we remove t first, we are missing a tile, hence t' cannot be blocked but has to be a cut tile in the remaining polyomino $P - t$. The presence of t preserves connectivity, i.e., $\{t, t'\}$ is a minimal cut on P . Because P has no holes, then t and t' must be diagonal neighbors, sharing the neighbors a and b . Furthermore, by definition neither of t and t' is blocked in some direction. We make a case distinction on the relation of these two directions.



- (a) If the unblocked directions of t and t' are orthogonal, one of the two adjacent tiles (w.l.o.g. a) cannot have any further neighbors. There can also be no tiles in the upper left corner, because the polyomino cannot cross the two free directions of t and t' (red marks).
- (b) If the unblocked directions of t and t' are parallel, there is only the tile c for which something can change if we remove t before t' .

■ **Figure 5** The red marks indicate that no tile is at this position; the dashed outline represents the rest of the polyomino.



- (a) If the removal direction of t is not crossed, the last blocking tile has to be convex (and has to be removed before).
- (b) If the removal direction of t crosses P , then P gets split into components A and B . Component B has a convex tile t' that needs to be removed before t .

■ **Figure 6** Polyominoes for which no convex tile should be removable, showing the contradiction to t being the first blocked convex tile in P getting removed.

The directions are orthogonal (Fig. 5a). Either a or b is a non-blocked convex tile, because t and t' are both non-blocked; w.l.o.g., let this be a . It is easy to see that independent of removing t or t' first, after removing a we can also remove the other one.

The directions are parallel (Fig. 5b). This case is slightly more involved. By assumption, we have a decomposition sequence beginning with t' . We show that swapping t' with our convex tile t in this sequence preserves feasibility.

The original sequence has to remove either a or b before it removes t , as otherwise the connection between the two is lost when t' is removed first. After either a or b is removed, t becomes a leaf and can no longer be important for connectivity. Thus, we only need to consider the sequence until either a or b is removed. The main observation is that a and b block the same tiles as t or t' , except for tile c as in Fig. 5b. However, when c is removed, it has to be a leaf, because a is still not removed and in the original decomposition sequence, t' has already been removed. Therefore, a tile $d \neq t'$ would have to be removed before c . Hence, the decomposition sequence remains feasible, concluding the proof. ◀

Next we show that such a convex tile always exists if the polyomino is decomposable.

► **Lemma 4.** *Let P be a decomposable polyomino. Then there exists a convex tile that is removable without destroying connectivity.*

Proof. We prove this by contradiction based on two possible cases.

Assume P to be a decomposable polyomino in which no convex tile is removable. Because P is decomposable, there exists some feasible decomposition sequence S . Let P_{convex} denote the set of convex tiles of P and let $t \in P_{\text{convex}}$ be the first removed convex tile in the decomposition sequence S . By assumption, t cannot be removed yet, so it is either blocked or a cut tile.

t is blocked. Consider the direction in which we would remove t . If it does not cut the polyomino, the last blocking tile has to be convex (and would have to be removed before t), see Fig. 6a. If it cuts the polyomino, the component cut off also must have a convex tile and the full component has to be removed before t , see Fig. 6b. This is again a contradiction to t being the first convex tile to be removed in S .

t is a cut tile. $P - t$ consists of exactly two connected polyominoes, P_1 and P_2 . It is easy to see that $P_1 \cap P_{\text{convex}} \neq \emptyset$ and $P_2 \cap P_{\text{convex}} \neq \emptyset$, because every polyomino of size $n \geq 2$ has at least two convex tiles of which at most one becomes non-convex by adding t . (A polyomino of size 1 is trivial.) Before being able to remove t , either P_1 or P_2 has to be completely removed, including their convex tiles. This is a contradiction to t being the first convex tile in S to be removed. \blacktriangleleft

3.2 An Efficient Algorithm

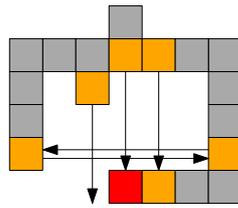
An iterative combination of these two lemmas proves the correctness of greedily removing convex tiles. As we show in the next theorem, using a search tree technique allows an efficient implementation of this greedy algorithm.

► **Theorem 5.** *A hole-free polyomino can be checked for decomposability/constructibility in time $O(N \log N)$.*

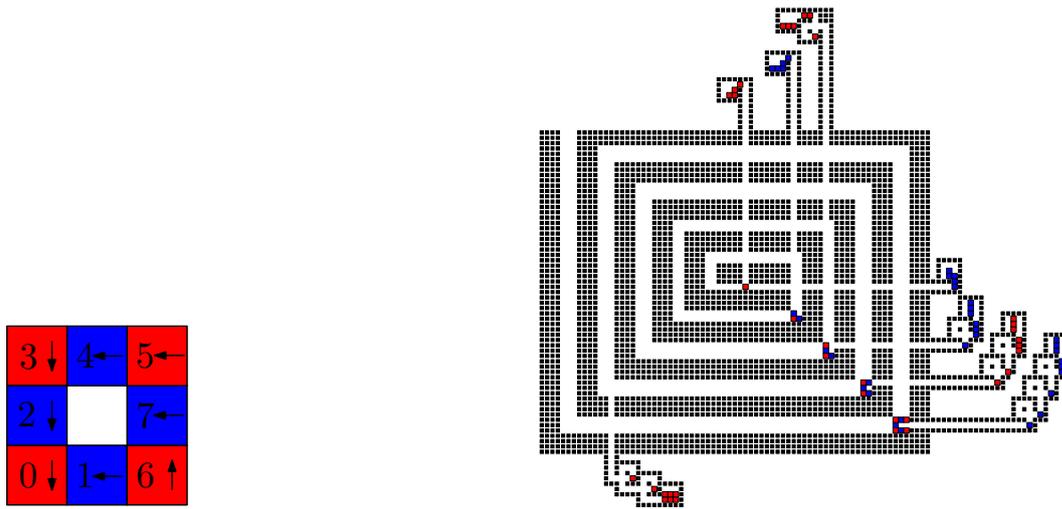
Proof. Lemma 3 allows us to remove any convex tile, as long as it is not blocked and does not destroy connectivity. Applying the same lemma on the remaining polyomino iteratively creates a feasible decomposition sequence. Lemma 4 proves that this is always sufficient. If and only if we can at some point no longer find a matching convex tile (to which we refer as *candidates*), the polyomino cannot be decomposable.

Let B be the time needed to check whether a tile t is blocked. A naïve way of doing this is to try out all tiles and check if t gets blocked, requiring time $O(N)$. With a preprocessing step, we can decrease B to $O(\log N)$ by using $O(N)$ binary search trees for searching for blocking tiles and utilizing that removing a tile can change the state of at most $O(1)$ tiles. For every vertical line x and horizontal line y going through P , we create a balanced search tree, i.e., for a total of $O(N)$ search trees. An x -search tree for a vertical line x contains tiles lying on x , sorted by their y -coordinate. Analogously define a y -search tree for a horizontal line y containing tiles lying on y sorted by their x -coordinate. We iterate over all tiles $t = (x, y)$ and insert the tile in the corresponding x - and y -search tree with a total complexity of $O(N \log N)$. Note that the memory complexity remains linear, because every tile is in exactly two search trees. To check if a tile at position (x', y') is blocked from above, we can simply search in the $(x' - 1)$ -, x' - and $(x' + 1)$ -search tree for a tile with $y > y'$. We analogously perform search queries for the other three directions, and thus have 12 queries of total cost $O(\log N)$.

We now iterate on all tiles and add all convex tiles that are not blocked and are not a cut tile to the set F (cost $O(N \log N)$). Note that checking whether a tile is a cut tile can be done in constant time, because it suffices to look into the local neighborhood. While F is not empty, we remove a tile from F , from the polyomino, and from its two search trees in



■ **Figure 7** When removing the red tile, only the orange tiles can become unblocked or convex.



■ **Figure 8** (Left) A polyomino P . Shown is the assembly order and the direction of attachment to the seed (tile 0). (Right) A maze environment for pipelined construction of the desired polyomino P . After the fourth cycle, each further cycle produces a new copy of P .

time $O(\log N)$. Next, we check the up to 12 tiles that are blocked first from the removed tile for all four orientations, see Fig. 7. Only these tiles can become unblocked or a convex tile. Those that are convex tiles, not blocked and no cut tile are added to F . All tiles behind those cannot become unblocked as the first tiles would still be blocking them. The cost for this is again in $O(\log N)$. This is continued until F is empty, which takes at most $O(N)$ loops each of cost $O(\log N)$. If the polyomino has been decomposed, the polyomino is decomposable/constructible by the corresponding tile sequence. Otherwise, there cannot exist such a sequence. By prohibiting to remove a specific tile, one can force a specific start tile. ◀

3.3 Pipelined Assembly

Given that a construction is always possible based on adding convex corners to a partial construction, we can argue that the idea of Manzoor et al. [15] for pipelined assembly can be realized for *every* constructible polyomino: We can transform the construction sequence into a spiral-shaped maze environment, as illustrated in Fig. 8. This allows it to produce D copies of P in $N + D$ cycles, implying that we only need $2N$ cycles for N copies. It suffices to use a clockwise order of four unit steps (west, north, east, south) in each cycle.

The main idea is to create a spiral in which the assemblies move from the inside to the outside. The first tile is provided by an initial south movement. After each cycle, ending

with a south movement, the next seed tile of the next copy of P is added. For every direction corresponding to the direction of the next tile added by the sequence, we place a tile depot on the outside of the spiral, with a straight-line path to the location of the corresponding attachment.

► **Theorem 6.** *Given a construction sequence $\sigma := ((d_1, l_1), \dots, (d_{N-1}, l_{N-1}))$ that constructs a polyomino P , we can construct a maze environment for pipelined tilt assembly, such that constructing D copies of P needs $O(N + D)$ unit steps. In particular, constructing one copy of P can be done in amortized time $O(1)$.*

A more detailed proof can be found in the full version of this paper [4].

4 Optimization Variants in 2D

For polyominoes that cannot be assembled, it is natural to look for a maximum-size subpolyomino that is constructible. As it turns out, this optimization variant is polyAPX-hard, i.e., we cannot hope for an approximation algorithm with an approximation factor within $\Omega(N^{\frac{1}{3}})$, unless $P = NP$.

► **Definition 7** (Maximum Tilt Assembly Problem). Given a polyomino P , the Maximum Tilt Assembly Problem (MAXTAP) asks for a sequence of tiles building a cardinality-maximal connected subpolyomino $P' \subseteq P$.

► **Theorem 8.** *MAXTAP is polyAPX-hard, even for tree-shaped polyominoes, and cannot be approximated within a factor of $\Omega(N^{\frac{1}{3}})$.*

The proof is based on a reduction from MAXIMUM INDEPENDENT SET (MIS) to MAXTAP. See full version [4] for details. On the positive side, we can give an $O(\sqrt{N})$ -approximation algorithm.

► **Theorem 9.** *The longest constructible path in a tree-shaped polyomino P is a \sqrt{N} -approximation for MAXTAP, and we can find such a path in polynomial time.*

Proof. Consider an optimal solution P^* and a smallest enclosing box B containing P^* . Then there must be two opposite sides of B having at least one tile of P^* . Consider the path S between both tiles. Because the area A_B of B is at least the number of tiles in P^* , $|S| \geq \sqrt{A_B}$ and a longest, constructible path in P has length at least $|S|$, we conclude that the longest constructible path is a \sqrt{N} -approximation.

To find such a path, we can search for every path between two tiles, check whether we can build this path, and take the longest, constructible path. ◀

Checking constructibility for $O(N^2)$ possible paths is rather expensive. However, we can efficiently approximate the longest constructible path in a tree-shaped polyomino with the help of *sequentially* constructible paths, i.e., the initial tile is a leaf in the final path.

► **Theorem 10.** *We can find a constructible path in a tree-shaped polyomino in $O(N^2 \log N)$ time that has a length of at least half the length of the longest constructible path.*

Proof. We only search for paths that can be built sequentially. Clearly, the longest such path is at least half as long as the longest path that can have its initial tile anywhere. We use the same search tree technique as before to look for blocking tiles. Select a tile of the polyomino as the initial tile. Do a depth-first search and for every tile in this search, check if it can be added to the path. If it cannot be added, skip all deeper tiles, as they also cannot

be added. During every step in the depth-first search, we only need to change a single tile in the search trees, doing $O(1)$ updates with $O(\log N)$ cost. As we only consider $O(N)$ vertices in the depth-first search, this results in a cost of $O(N \log N)$ for a fixed start tile. It is trivial to keep track of the longest such constructible path. Repeating this for every tile results in a running time of $O(N^2 \log N)$. ◀

In tree-shaped polyominoes, finding a constructible path is easy. For simple polyominoes, additional arguments and data structures lead to a similar result.

► **Theorem 11.** *In simple polyominoes, finding the longest of all shortest paths that are sequentially constructible takes $O(N^2 \log N)$ time.*

See full version [4] for details.

5 Three-Dimensional Shapes

An interesting and natural generalization of TAP is to consider three-dimensional shapes, i.e., polycubes. As it turns out, the local considerations for simply connected two-dimensional shapes are no longer sufficient. In the following we show that deciding whether a polycube is constructible is NP-hard. Moreover, it is NP-hard to check whether there is a constructible path from a start cube s to an end cube t in a partial shape.

As a stepping stone, we start with a restricted version of the three-dimensional problem.

► **Theorem 12.** *It is NP-hard to decide if a polycube can be built.*

The proof is based on a reduction from 3SAT and omitted for lack of space; see full version [4] for details.

The difficulties of construction in 3D are highlighted by the fact that even identifying constructible connections between specific positions is NP-hard.

► **Theorem 13.** *It is NP-hard to decide whether a path from one tile to another can be built in a general polycube.*

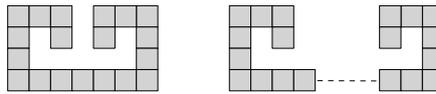
The proof proceeds by a reduction from SAT; see full version [4] for details.

6 Conclusion/Future Work

We have provided a number of algorithmic results for Tilt Assembly. Various unsolved challenges remain. What is the complexity of deciding TAP for non-simple polyominoes? While Lemma 4 can be applied to all polyominoes, we cannot simply remove any convex tile. Clearly, $\text{TAP} \in \text{NP}$ if the polyomino is encoded tile by tile, which is usually done in practice. Can we find a constructible path in a polyomino from a given start and endpoint? This would help in finding a \sqrt{N} -approximation for non-simple polyominoes. How can we optimize the total makespan for constructing a shape? And what options exist for non-constructible shapes?

An interesting approach may be to consider *staged* assembly, as shown in Fig. 9, where a shape gets constructed by putting together subpolyominoes, instead of adding one tile at a time. This is similar to staged tile self-assembly [10, 11]. This may also provide a path to sublinear assembly times, as a hierarchical assembly allows massive parallelization. We conjecture that a makespan of $O(\sqrt{N})$ for a polyomino with N tiles can be achieved.

All this is left to future work.



■ **Figure 9** (Left) A polyomino that cannot be constructed in the basic TAP model. (Right) Construction in a staged assembly model by putting together subpolyominoes.

References

- 1 Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Golnaz Habibi, and James McLurkin. Reconfiguring massive particle swarms with limited, global control. In *Proc. Int. Symp. Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS)*, pages 51–66, 2013.
- 2 Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, and James McLurkin. Particle computation: Designing worlds to control robot swarms with only global signals. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 6751–6756, 2014.
- 3 Aaron T. Becker, Chris Ertel, and James McLurkin. Crowdsourcing swarm manipulation experiments: A massive online user study with large swarms of simple robots. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2825–2830, 2014.
- 4 Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, Christian Rieck, Christian Scheffer, and Arne Schmidt. Tilt Assembly: Algorithms for Micro-Factories that Build Objects with Uniform External Forces. *arXiv preprint arXiv:1709.06299*, 2017.
- 5 Aaron T. Becker, Ouajdi Felfoul, and Pierre E. Dupont. Simultaneously powering and controlling many actuators with a clinical MRI scanner. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pages 2017–2023, 2014.
- 6 Aaron T. Becker, Ouajdi Felfoul, and Pierre E. Dupont. Toward tissue penetration by MRI-powered millirobots using a self-assembled Gauss gun. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 1184–1189, 2015.
- 7 Aaron T. Becker, Golnaz Habibi, Justin Werfel, Michael Rubenstein, and James McLurkin. Massive uniform manipulation: Controlling large populations of simple robots with a common input signal. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pages 520–527, 2013.
- 8 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert Schweller, Scott M. Summers, and Andrew Winslow. Two hands are better than one (up to constant factors). In *Proc. Int. Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 172–184, 2013.
- 9 Ho-Lin Chen and David Doty. Parallelism and time in hierarchical self-assembly. *SIAM Journal on Computing*, 46(2):661–709, 2017.
- 10 Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. *Natural Computing*, 7(3):347–370, 2008.
- 11 Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Computer Science*, 671:4–18, 2017.
- 12 Paul Seung Soo Kim, Aaron T. Becker, Yan Ou, Anak Agung Julius, and Min Jun Kim. Imparting magnetic dipole heterogeneity to internalized iron oxide nanoparticles for microorganism swarm control. *Journal of Nanoparticle Research*, 17(3):1–15, 2015.
- 13 Paul Seung Soo Kim, Aaron T. Becker, Yan Ou, Min Jun Kim, et al. Swarm control of cell-based microrobots using a single global magnetic field. In *Proc. Int. Conf. Ubiquitous Robotics and Ambient Intelligence (URAI)*, pages 21–26, 2013.

- 14 Arun V. Mahadev, Dominik Krupke, Jan-Marc Reinhardt, Sándor P. Fekete, and Aaron T. Becker. Collecting a swarm in a grid environment using shared, global inputs. In *Proc. IEEE Int. Conf. Autom. Sci. and Eng. (CASE)*, pages 1231–1236, 2016.
- 15 Sheryl Manzoor, Samuel Sheckman, Jarrett Lonsford, Hoyeon Kim, Min Jun Kim, and Aaron T. Becker. Parallel self-assembly of polyominoes under uniform control inputs. *IEEE Robotics and Automation Letters*, 2(4):2040–2047, 2017.
- 16 Hamed Mohtasham Shad, Rose Morris-Wright, Erik D. Demaine, Sándor P. Fekete, and Aaron T. Becker. Particle computation: Device fan-out and binary memory. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 5384–5389, 2015.
- 17 Shiva Shahrokhi and Aaron T. Becker. Stochastic swarm control with global inputs. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pages 421–427, 2015.
- 18 Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.