


Efficient Shape Formation by 3D Hybrid Programmable Matter: An Algorithm for Low Diameter Intermediate Structures

Kristian Hinnenthal ✉ 🏠 

Paderborn University, Germany

David Liedtke ✉ 🏠 

Paderborn University, Germany

Christian Scheideler ✉ 🏠 

Paderborn University, Germany

Abstract

This paper considers the shape formation problem within the 3D hybrid model, where a single agent with a strictly limited viewing range and the computational capacity of a deterministic finite automaton manipulates passive tiles through pick-up, movement, and placement actions. The goal is to reconfigure a set of tiles into a specific shape termed an *icicle*. The icicle, identified as a dense, hole-free structure, is strategically chosen to function as an intermediate shape for more intricate shape formation tasks. It is designed for easy exploration by a finite state agent, enabling the identification of tiles that can be lifted without breaking connectivity. Compared to the line shape, the icicle presents distinct advantages, including a reduced diameter and the presence of multiple removable tiles. We propose an algorithm that transforms an arbitrary initially connected tile structure into an icicle in $\mathcal{O}(n^3)$ steps, matching the runtime of the line formation algorithm from prior work. Our theoretical contribution is accompanied by an extensive experimental analysis, indicating that our algorithm decreases the diameter of tile structures on average.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Programmable Matter, Shape Formation, 3D Model, Finite Automaton

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.15

Related Version *Full Version*: 10.48550/arXiv.2401.17734 [15]

Funding This work was supported by the DFG Project SCHE 1592/10-1.

1 Introduction

Advancements in molecular engineering have led to the development of a series of computing DNA robots designed for nano-scale operations. These robots are intended to perform simple tasks such as transporting cargo, facilitating communication, navigating surfaces of membranes, and pathfinding [29, 1, 22, 4]. Envisioning the future of nanotechnology, we anticipate a scenario where a collective of computing particles collaboratively acts as programmable matter – a homogeneous material capable of altering its shape and physical properties programmably. There are numerous potential applications: For environmental remediation, particles may construct nanoscale filtration systems to remove pollutants from air or water. They may also be deployed within the human body to construct intricate structures for targeted drug delivery, perform nanoscale surgeries, or repair damaged tissues at a cellular level. Additionally, they could assemble nanoscale circuits and components, enabling the development of more efficient and compact electronic devices. Each of those scenarios is an application of the *shape formation problem*, which is the subject of this paper.



© Kristian Hinnenthal, David Liedtke, and Christian Scheideler;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 15; pp. 15:1–15:20

Leibniz International Proceedings in Informatics



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

Over the past few decades, various models of programmable matter have emerged, primarily distinguished by the activity of entities within them. Passive systems consist of entities (called tiles) that undergo movement and bonding exclusively in response to external stimuli, such as current or light, or based on their inherent structural properties, such as specific glues on the surfaces of tiles. Examples of these passive systems include the DNA tile assembly models aTAM, kTAM, and 2HAM, which are extensively discussed in the survey [25], as well as population protocols [2], and slime molds [5]. In contrast, active systems consist of entities (called particles, agents or robots) that independently perform computation and movement to accomplish tasks. Notable examples encompass the Amoebot model [7], modular self-reconfigurable robots [27, 30], the nubot model [34], metamorphic robots [6, 31], and swarm robotics [33].

While fabricating computing DNA robots remains challenging, producing simple passive tiles from folded DNA strands is efficient and scalable [14]. The hybrid model of programmable matter [12, 13, 16, 23, 19] offers a compromise between feasibility and utility. This model involves a small number of active agents with the computational capabilities of deterministic finite automata together with a large set of passive building blocks, called tiles. Agents can manipulate the structure of tiles by picking up a tile, moving it, and placing it at some spot. A key advantage of the hybrid approach lies in the reusability of agents upon completing a task, where in purely active systems, particles become part of the formed structure.

In this paper, we address the shape formation problem within the 3D hybrid model, with the ultimate goal of transforming an arbitrary initial arrangement of tiles into a predefined shape. We consider tiles in the shape of rhombic dodecahedra, i.e., polyhedra featuring 12 congruent rhombic faces, positioned at nodes within the adjacency graph of face-centered cubic (FCC) stacked spheres (see Figure 1a). Unlike rectangular tiles, the rhombic dodecahedron presents a distinct advantage: it allows an agent to orbit around a tile without risking connectivity. This property is particularly valuable in liquid or low gravity environments, where it prevents unintended separation between the agents and the tiles.

Achieving universal 3D shape formation faces a key challenge: identifying tiles that can be lifted without disconnecting the tile structure (referred to as *removable* tiles). Even if such tiles exist, locating them requires exploring the tile structure, demanding $\Omega(D \log(\Delta))$ memory bits for graphs with a diameter D and degree Δ [11]. When limited to constant memory, navigating plane labyrinths requires two placeable markers (pebbles) [17, 3]. In the 2D context, finding removable tiles is impossible without prior modification of the tile structure, as discussed in [13]. In 3D, complexity increases significantly, with instances where any tile movement can locally disconnect the structure. As discussed above, the agent is unable to verify whether this disconnection also occurs globally. To address these challenges, we make the assumption that the agent carries a tile initially, using it to uncover removable tiles through successive tile movements. It is still entirely unclear whether otherwise a removable tile can be found in all 3D instances. For that reason, our primary goal is to construct an intermediate structure that is easily navigable by constant-memory agents and allows the identification of removable tiles without relying on an initially carried tile.

1.1 Our Contribution

The intermediate structure we propose is termed an *icicle*, characterized by a platform representing a parallelogram and downward-extending lines of tiles from the platform (see Figure 1a). We present a single-agent algorithm that transforms any initially connected tile structure into an icicle in $\mathcal{O}(n^3)$ steps, matching the efficiency of the line formation algorithm from prior work [16]. While both the icicle and the line enable agents without an

initial tile to find removable tiles, the icicle presents distinct advantages. In the best-case scenario, the diameter D of an icicle can be as low as $\mathcal{O}(n^{\frac{1}{3}})$, whereas a line consistently maintains a diameter of n . Furthermore, an icicle encompasses multiple removable tiles, which removes the necessity to traverse the intermediate shape completely to locate a removable tile. Our paper includes comprehensive simulation results, indicating that, on average, our algorithm reduces the diameter of the tile structure. In addition, the runtime observed in the simulations consistently falls below the bound established in our runtime analysis. Across all simulations, the runtime remains well within the vicinity of n^2 . It is noteworthy that we identified an edge case where the diameter could increase by a factor of $\mathcal{O}(n^{\frac{1}{3}})$, although we believe this to be the worst-case.

1.2 Related Work

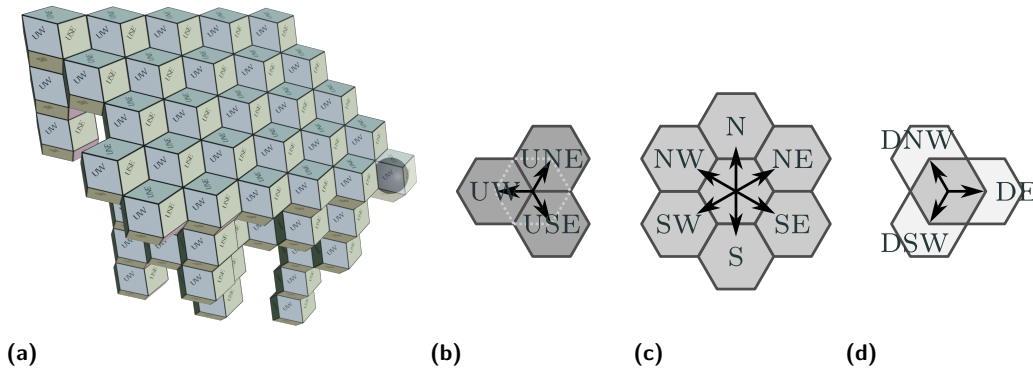
The 3D variant of the hybrid model was introduced in [16], where the authors presented an algorithm capable of transforming any connected input configuration into a line in $\mathcal{O}(n^3)$ steps. In [19], the authors address the coating problem, providing a solution that solves the problem in worst-case optimal $\mathcal{O}(n^2)$ steps. They assume a single active agent that has access to a constant number of distinguishable tile types.

Significant progress has been made in recent years regarding the 2D version of the hybrid model. For instance, in [13], the authors address the 2D shape formation problem, presenting algorithms for a single active agent that efficiently constructs line, block, and tree structures - each being hole-free structures with specific advantages and disadvantages - in worst-case optimal $\mathcal{O}(n^2)$ steps. Another publication, [12], explores the recognition of parallelograms with a specific height-to-length ratio. The most recent publication [23] solves the problem of maintaining a line of tiles in presence of multiple agents and dynamic failures of the tiles.

Closely tied to the hybrid model is the well-established Amoebot model, where computing particles traverse an infinite triangular lattice through expansions and contractions. In [8], the authors showcase the construction of simple shapes like hexagons or triangles within the Amoebot model. Expanding on this work, [9] introduces a universal shape formation algorithm capable of constructing an arbitrary input shape using a constant number of equilateral triangles, with the scale depending on the number of amoebots. Notably, this work assumes common chirality, a sequential activation schedule, and randomization. Subsequent improvements are presented in [10], where a deterministic algorithm is introduced, enabling amoebots to form any Turing-computable shape without the need for common chirality or randomization. In [20], the authors consider shape formation in the presence of a finite number of faults, where a fault resets an amoebot's memory. They solve the hexagon formation problem, assuming the existence of a fault-free leader. A recent extension of the Amoebot model, discussed in [24], considers joint movements of Amoebots. The authors simulate various shape formation algorithms as a proof of concept.

In both [13] and this paper, shape formation algorithms are introduced that construct an intermediate shape, intended to serve as the foundation for more advanced shape formation algorithms. A similar strategy is explored in [18], where 2D lattice-based modular robots initially transform into a canonical shape before achieving the final desired shape. An approach that does not rely on canonical intermediate structures is considered in [26]. The authors present primitives for the Amoebot model that establish shortest path trees within the amoebot structure and subsequently directly route amoebots to their target position.

The concept of shape formation is extensively studied in the field of modular robotics and metamorphic robots, often referred to as self-reconfiguration. A comprehensive survey on this topic can be found in [28]. In the field of swarm robotics, shape formation is often closely related to the problem of computing collision-free paths [32, 21].



■ **Figure 1** (a) An example configuration that has the shape of an icicle; the agent (depicted as a sphere) is positioned at a tiled node within the platform representing a parallelogram. (b–d) The twelve compass directions divided into upwards (b), plane (c) and downwards directions (d).

1.3 Model Definition

We consider a single active agent r with limited sensing and computational power that operates on a finite set of passive *tiles* positioned at nodes of some specific underlying graph G , which we define in the following. Consider the close packing of equally sized spheres at each point of the infinite face-centered cubic lattice. Let $G = (V, E)$ be the adjacency graph of spheres in that packing, and consider an embedding of G in \mathbb{R}^3 in which all edges have equal length, e.g., the trivial embedding where the edge length equals the radius of the spheres. Cells in the dual graph of G w.r.t. that embedding have the shape of rhombic dodecahedra, i.e., polyhedra with 12 congruent rhombic faces (see Figure 1a). This is also the shape of every cell in the Voronoi tessellation of G , i.e., that shape completely tessellates 3D space. Consider a finite set of tiles that have the shape of rhombic dodecahedra. Tiles are passive, in the sense that they cannot perform any computation or movement on their own. A node $v \in V$ is *tiled*, if there is a passive tile positioned at v ; otherwise node v is *empty*. Each node can hold at most one tile and each tile is placed at at most one node at a time. Each node in V has precisely twelve neighbors whose relative positions are described by the twelve compass directions UNE, UW, USE, N, NW, SW, S, SE, NE, DNW, DSW and DE (see Figures 1b–1d). Take note that G contains infinitely many copies of the infinite triangular lattice, which serves as the underlying graph in the 2D variant. This allows us to visually depict 3D examples as a stack of 2D hexagonal tiles, as shown in Figure 1.

A *configuration* $C = (\mathcal{T}, p)$ is the set \mathcal{T} that contains all tiled nodes together with the agent's position p . We call C *connected*, if $G|_{\mathcal{T}}$ is connected or if $G|_{\mathcal{T} \cup \{p\}}$ is connected and the agent carries a tile, where $G|_W$ denotes the subgraph of G induced by some nodeset W . That is, we allow the subgraph induced by all tiled nodes to disconnect, as long as a tile carried by the agent maintains connectivity. This constraint prevents the agent and tiles to drift apart, e.g., in liquid or low gravity environments.

The agent r is the only active entity in this model. It has strictly limited sensing and computing power and can act on passive tiles by picking up a tile, moving and placing it at some spot. We assume that tiles can be moved through other tiles, e.g., by Particularly, we assume an agent with the computational capabilities of a deterministic finite automaton that performs discrete steps of *Look-Compute-Move* cycles. In the *look*-phase, the agent observes whether its current position p and the twelve neighbors of p are tiled or empty. The agent is equipped with a compass that allows it to distinguish the relative positioning of

its neighbors using the twelve above mentioned compass directions. Its initial rotation and chirality can be arbitrary, but we assume that it remains consistent throughout the execution. For ease of presentation, our algorithms and their analysis are described according to the robot’s local view, i.e., we do not distinguish between local and global compass directions. Based on the information gathered in the look phase, the agent determines its next state transition according to the finite automaton in the *compute*-phase. In the *move* phase, the agent performs an *action* that corresponds to the prior state transition. It either (i) moves to an empty or tiled node adjacent to p , (ii) places a tile at p , if $p \notin \mathcal{T}$ and r carries a tile, (iii) picks up a tile from p , if $p \in \mathcal{T}$ and r carries no tile, or (iv) terminates. The agent can carry at most one tile at a time and during actions (ii) and (iii) the agent loses and gains a tile, respectively. It’s worth noting that we allow the agent to move through tiles while carrying one simultaneously. From a practical standpoint, this capability can be facilitated by conceptualizing tiles as hollow and foldable. It is assumed that the agent is initially positioned at a tiled node, as otherwise, there might be no valid action available. Additionally, we assume that the agent initially carries a tile, a justification for which was provided in Section 1. While the agent is technically a finite automaton, we describe algorithms from a higher level of abstraction textually and through pseudocode. It is easy to see that a constant number of variables of constant-size domain each can be incorporated into the agent’s constantly many states.

1.4 Problem Statement

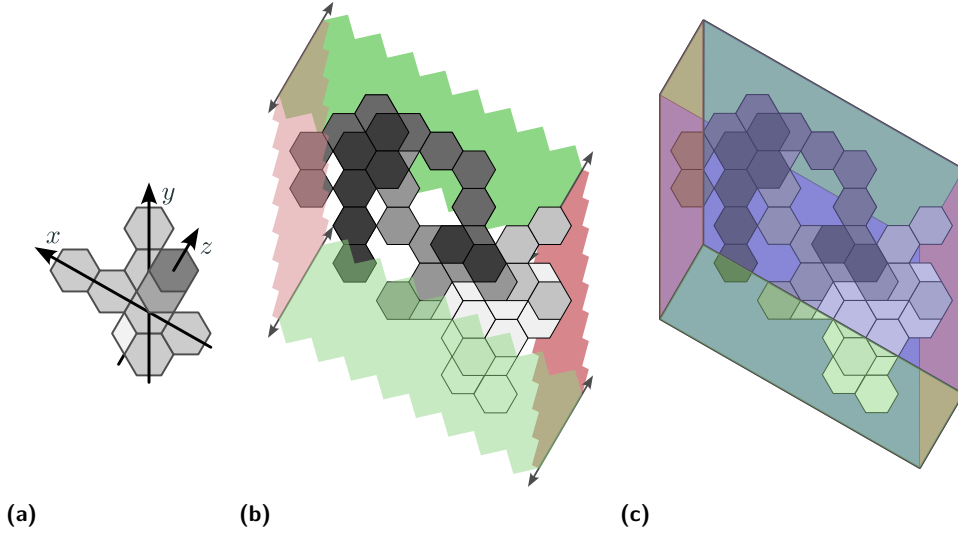
Consider an arbitrary initially connected configuration $C^0 = (\mathcal{T}^0, p^0)$ with $p^0 \in \mathcal{T}$. Superscripts in our notation generally refer to step numbers and may be omitted if they are clear from the context. An algorithm solves the *icicle formation problem*, if its execution results in a sequence of connected configurations $C^0 = (\mathcal{T}^0, p^0), \dots, C^T = (\mathcal{T}^T, p^T)$ such that nodes in \mathcal{T}^T are in the shape of an icicle (which we define below), C^t results from C^{t-1} for $1 \leq t \leq T$ by applying some action (i)–(iii) to p^{t-1} , and the agent terminates (iv) in step T .

For some node $v \in V$, we denote $v + x$ the node that is neighboring v in some compass direction x and $-x$ the opposite compass direction of x , e.g., $-UNE = DSW$. We call a maximal consecutive array of tiles in direction N and S a *column*, in direction NW and SE a *row*, and in direction UNE and DSW a *tower*. A *parallelogram* is a maximal consecutive array of equally sized columns c_0, \dots, c_m (ordered from west to east) whose southernmost tiles at nodes v_0, \dots, v_m are contained in the same row, i.e., $v_i + SE = v_{i+1}$ for all $0 \leq i < m$. In a *partially filled* parallelogram, column c_0 can have smaller size than columns c_1, \dots, c_m .

An *icicle* is defined as a connected set of towers whose uppermost tiles are contained within the same (partially filled) parallelogram, as illustrated in Figure 1a. In other words, tiles “grow” from a single uppermost parallelogram in the DSW direction, hence the chosen name “icicle”. Notably, in an icicle, any tile with a neighboring tile at UNE but not at DSW (some locally DSW -most tile below the parallelogram) can be picked up without violating connectivity (it is *removable*). If there is no such tile, i.e., all towers have size one, the northernmost tile of the westernmost column is removable.

1.5 Structure of the Paper

In Section 2, we introduce all essential terminology. Following that, we present a non-halting icicle-formation algorithm in Section 3, prove that it converges any initially connected configuration into an icicle in Section 4, and provide its termination criteria and runtime analysis in Section 5. Finally, we discuss the results obtained from simulation in Section 6.



■ **Figure 2** Illustrating the x -, y -, and z -coordinate axes (a), the bounding cylinder (b), which infinitely extends in directions UNE and DSW as indicated by the arrows, and the bounding box (c) of an example configuration. For ease of distinction, tiles are shaded according to their z -coordinate, with brighter shades representing lower z -coordinates. In the example, there is one layer that contains two fragments (darkest shade of gray), and four layers that each contain a single fragment.

2 Preliminaries

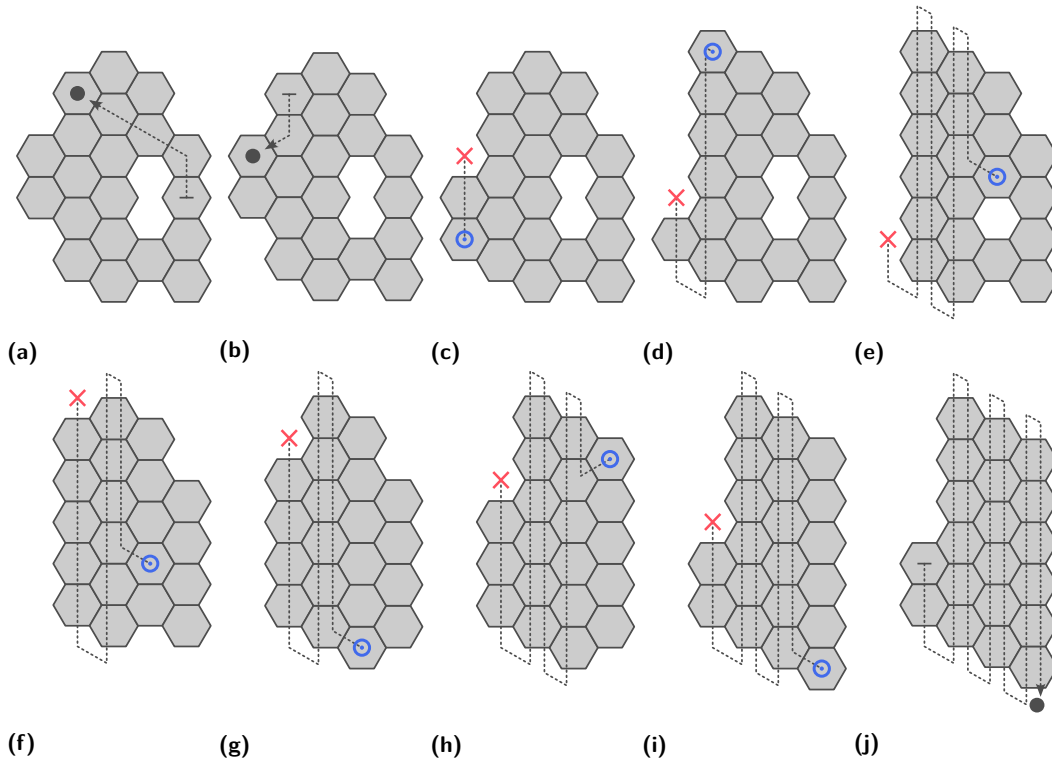
We assign x, y and z coordinates to each node $v \in V$, denoted by $c(v) = (x(v), y(v), z(v))$, where the x -coordinates grow from SE to NW, y -coordinates from S to N, and z -coordinates from DSW to UNE (see Figure 2a). The coordinates transition between neighbors as follows:

► **Observation 1.** Let w be some reference node with $c(w) = (0, 0, 0)$. The following holds:

$$\begin{array}{lll}
 c(w + \text{UNE}) = (0, 0, 1) & c(w + \text{UW}) = (1, -1, 1) & c(w + \text{USE}) = (0, -1, 1) \\
 c(w + \text{N}) = (0, 1, 0) & c(w + \text{NW}) = (1, 0, 0) & c(w + \text{SW}) = (1, -1, 0) \\
 c(w + \text{S}) = (0, -1, 0) & c(w + \text{SE}) = (-1, 0, 0) & c(w + \text{NE}) = (-1, 1, 0) \\
 c(w + \text{DSW}) = (0, 0, -1) & c(w + \text{DE}) = (-1, 1, -1) & c(w + \text{DNW}) = (0, 1, -1)
 \end{array}$$

Given some nodeset S , let x_{min}^S, x_{max}^S be the minimum and maximum x -coordinate of any node in S , and define $y_{min}^S, y_{max}^S, z_{min}^S$ and z_{max}^S accordingly. We normalize coordinates according to the minimum coordinates in the initial set of tiled nodes \mathcal{T}^0 , i.e., we set $x_{min}^{\mathcal{T}^0} = y_{min}^{\mathcal{T}^0} = z_{min}^{\mathcal{T}^0} = 0$. The *bounding cylinder* $\mathfrak{C}(S)$ is the set of all nodes (both empty and tiled) whose coordinates are bounded by the minimum and maximum x - and y -coordinates in S , i.e., $\mathfrak{C}(S) = \{v \in V \mid x_{min}^S \leq x(v) \leq x_{max}^S, y_{min}^S \leq y(v) \leq y_{max}^S\}$ (see Figure 2b). Similarly, in the *bounding box* $\mathfrak{B}(S)$ we further bound by the z -coordinate, i.e., $\mathfrak{B}(S) = \{v \in \mathfrak{C}(S) \mid z_{min}^S \leq z(v) \leq z_{max}^S\}$. We refer to the extent of a bounding box along the x -, y - and z -axes as its *width*, *height*, and *depth*. Note that by the choice of our coordinate axes, the bounding box is always a filled (potentially degenerated) parallelepiped (a 3D rhomboid; see Figure 2c). A node v is *inside* the bounding cylinder (box) of S , if $v \in \mathfrak{C}(S)$ ($v \in \mathfrak{B}(S)$); otherwise, v is *outside* of the bounding cylinder (box) of S .

A *layer* L_i is the set of all nodes with z -coordinate i that are contained in the bounding cylinder of all tiled nodes, i.e., $L_i = \{v \in \mathfrak{C}(\mathcal{T}) \mid z(v) = i\}$. We refer to nodes with z -coordinate greater than and less than i as the nodes *above* and *below* layer L_i , respectively. The nodeset of a connected component of $G|_{L_i \cap \mathcal{T}}$ is called a *fragment (of L_i)* (see Figure 2).



■ **Figure 3** The parallelogram formation algorithm on a 2D configuration. The agent performs multiple steps between each depicted configuration. In (a) and (b) the agent finds a westernmost column, and in (j) the agent terminates. In all other cases, a tile is shifted from the cross to the circle, where the dashed lines indicate the path traversed before placing the tile. The path back to where the tile is picked up as well as the movement to the next column (e.g., (e)–(f)) is not shown.

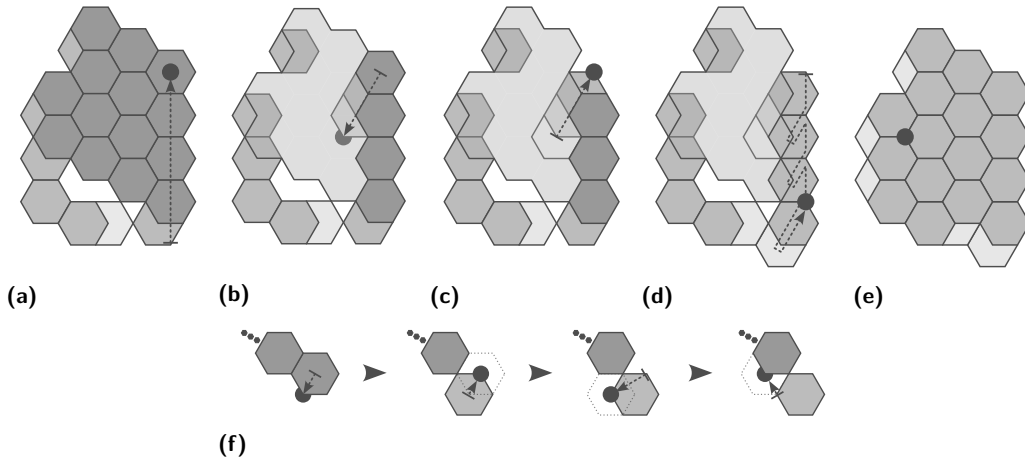
3 The Algorithm

From a high-level perspective, the agent iteratively transforms locally uppermost fragments into partially filled parallelograms. This involves rearranging tiles within the same layer and, at times, positioning tiles below the current layer to ensure connectivity. Whenever the agent encounters tiles of some layer above, it moves further upwards. Once a parallelogram is successfully formed, the subsequent step entails its *projection*. Essentially, during this projection, each tile in the fragment is shifted to the first empty node in the DSW direction.

In the following, we provide detailed textual descriptions of the parallelogram formation and projection procedures `BUILDPAR` and `PROJECT`, as well as the full icicle formation algorithm `BUILDICICLE`. For completeness, their pseudocodes can be found in Appendix A.

3.1 A 2D Parallelogram Formation Algorithm

Refer to Figure 3 for an illustrative example of the algorithm in action. The algorithm initiates with the agent searching for a locally westernmost column. In configurations where multiple columns share the same x -coordinate and are locally westernmost, the agent prioritizes finding the northernmost among them. This is achieved by moving in the NW, SW, and N directions, prioritized in that order, until no more tile is encountered in any of these directions. Eventually the agent stops upon reaching the northernmost tiled node v of some column c . We refer to the steps involved in finding column c as the *search phase*.

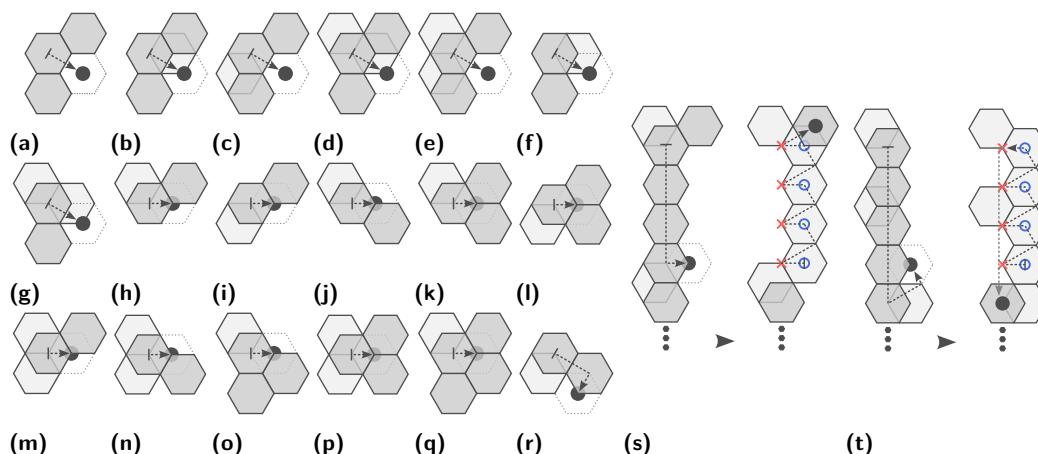


■ **Figure 4** During a projection, the agent (black disk) shifts each tile of a fragment in direction DSW. Detailed in (a-d) is the projection of a single column; (e) is a snapshot of the configuration after the projection. The special case of a parallelogram with a height of one is shown in (f). To maintain connectivity in that case, the agent moves SW + DNW to transition below the next column.

Subsequently, it executes the BUILDPAR procedure, which we describe in the following: Starting from node v , the agent traverses each column in the configuration from N to S. If, during the traversal of the first column c , the agent encounters either a more western column (as depicted in Figure 3b) or a column with the same x -coordinate as c to the north while moving N in the next column c' , it discontinues the current traversal and transitions to the search phase. Notably, in the latter case, it first fully traverses column c' in direction N and afterwards moves to the first column west of c' . This technical detail will play an important role in the runtime analysis. While traversing a column in the S direction, the agent actively looks for an empty node that *violates* the shape of a (partially filled) parallelogram with westernmost column c . Specifically, it checks the two empty nodes immediately above (excluding column c) and below each column, as well as each empty neighbor to the east of the column. Upon finding such a violating empty node w , the agent first places its carried tile at w and then returns to column c to retrieve the tile from v . Subsequently, this exchange of tiles is termed as a *tile shift* from v to w or as *shifting* (the tile) from v to w (recall that the agent initially carries a tile that was never placed at any node). After picking up the tile at v , the agent moves to an adjacent tile and transitions to the search phase again. The agent terminates at the empty node S of the easternmost column once the configuration is fully traversed without encountering any violating nodes. Any of the following conditions are sufficient for an empty node w to be considered violating: (1) w has a tile at N, NE and SE (e.g., Figure 3c), (2) w has a tile at S and SE (e.g., Figure 3d) and is not N of the westernmost column (recall that we allow the parallelogram to be partially filled), (3) w has a tile at NW and N (e.g., Figures 3e–3g and 3i), (4) w has a tile at NW, SW and S (e.g., Figure 3h).

3.2 An Icicle Formation Algorithm

From a high-level perspective, the construction of an icicle involves the iterative transformation of a locally uppermost fragment into a parallelogram, followed by a projection of the fragment in the DSW direction. When applying the parallelogram construction algorithm in a 2D configuration, the agent can always shift the tile at the northernmost node v of a locally westernmost column without violating connectivity (Figure 5a illustrates that connectivity



■ **Figure 5** Illustrating all scenarios in which the northernmost node of a locally westernmost column is not removable. For brevity, (a–r) only illustrate the agent’s movement (indicated by arrows) to the empty node (with a dashed outline) that is tiled next; (s) and (t) also portray the subsequent tile shifts. In (r), the agent may alternatively enter BUILDPAR if the outlined node were tiled. Note that (s) and (t) only show instances where a tile at DSW (s) and DE (t) is encountered.

is preserved in the only critical 2D case). In a 3D configuration, the situation becomes more intricate. There are multiple cases in which the tile at v must remain in its immediate neighborhood to avoid violating connectivity. Additionally, there is a case in which the tile at v cannot be moved at all unless neighboring tiles are also moved. We categorize these cases based on specific properties of node v , which we define as follows:

► **Definition 1.** Let $v \in \mathcal{T}$ be an arbitrary tiled node. Denote by $N(v)$ the neighborhood of v (excluding v), and by $N_{\mathcal{T}}(v)$ its subset of only tiled nodes. Node v is *removable*, if the tiled neighbors of v are locally connected, i.e., $G|_{N_{\mathcal{T}}(v)}$ is connected. Node v is *shiftable*, if $G|_{N_{\mathcal{T}}(v)}$ is disconnected and there exists a node $w \neq v$ (termed *bridge node of v*) for which $G|_{N_{\mathcal{T}}(v) \cup \{w\}}$ is connected. Any node that is neither removable nor shiftable is termed *unmovable*.

We now state the full icicle algorithm: The agent starts in the search phase where it repeatedly moves UW, USE, UNE, NW, SW and N until it eventually stops at some node v .

If node v is removable, the parallelogram traversal procedure BUILDPAR is entered. There are three possible outcomes: the agent returns from the procedure after finding a more western column or some tile above, after placing a tile, or at the empty node s of the fragment’s easternmost column. In the first case, the agent transitions to the search phase. In the second case, the agent first moves back to pick up the tile at node v , then moves to the next tile at s or SE , and afterwards transitions to the search phase. In the third case, the current fragment forms a correctly shaped parallelogram and the agent proceeds by executing the PROJECT procedure. During PROJECT, each tile of the fragment is projected in the DSW direction. Starting with the easternmost column, tiles are projected columnwise from east to west and within the columns from N to S (see Figures 4a–4e). Let v_0, \dots, v_k be the nodes of the currently projected column ordered from N to S . For each $i = 0, \dots, k$, the agent performs a tile shift from v_i to the first empty node w_i in direction DSW of v_i . After picking up the last tile of the column at v_k , the agent moves NW and continues the projection in the western neighboring column. In the special case of a degenerated parallelogram with a height of one, after picking up a tile, the agent moves SW and DNW instead (see Figure 4f). These additional steps ensure that connectivity is maintained during the projection. Once the last tile of the fragment is projected, the agent transitions to the search phase in the layer below.

15:10 Efficient Shape Formation by 3D Hybrid Programmable Matter

Otherwise, if the agent stops at a non-removable node v , it acts according to the case distinction outlined below, prioritized in the given order (refer to Figure 5 for a graphical overview). Subsequently, the agent transitions to the search phase, concluding our algorithm.

- ▶ **Case A.** If $v + SE$ is a bridge node of v (thereby v is shiftable and $v + SE$ is empty), then shift the tile from v to $v + SE$ (see Figures 5a–5g), and move to $v + S$ afterwards.
- ▶ **Case B.** If $v + DE$ is a bridge node of v , and at least one neighboring tile is not at DSW or SE , then shift the tile from v to $v + DE$ (see Figures 5h–5q), and move to the first tile at $v + S$, $v + SE$ or $v + NE$. Additionally, if $v + S$ is empty and both $v + SE$ and $v + NE$ are tiled, then traverse the next column starting at $v + SE$ in direction S . If during that traversal a tile at UW , USE or SW is encountered, then immediately transition to the search phase.
- ▶ **Case C.** If the only tiled neighbors of v are at DSW and SE , then move SE and observe node $w = v + SE + DSW$. If w is empty, then shift the tile from v to w (see Figure 5r), and move to $v + SE$ afterwards. Otherwise, if w is already tiled, move back to v and enter `BUILDPAR`.
- ▶ **Case D.** If the only tiled neighbors of v are at DNW , S and NE (v is unmovable, see Figure 5s), then follow these steps: First, move S until some node w is entered that has a neighboring tile at UW , USE , SW , DSW , SE or DE , or until there is no more tile in direction S . If w has a tile at UW , USE or SW , then immediately transition to the search phase. Otherwise, shift each tile in the column that is somewhere N of w in direction DE (including w if $w + DE$ is empty). To be precise, let $v_k, v_{k-1}, \dots, v_1, v$ be the nodes of the column ordered from S to N starting at $v_k = w + N$ (or $v_k = w$ if $w + DE$ is empty). Perform a tile shift from v_i to $v_i + DE$ for each i with $k \geq i > 0$. After the tile shift at v_i with $i > 0$, move $NE + DNW$ to be positioned at $v_{i-1} + DE$ (to preserve connectivity). Once the final tile is picked up, move to $v + NE$.
- ▶ **Case E.** If the only tiled neighbors of v are at DNW and S (see Figure 5t), then proceed analogously to the previous case, with the exception that tiles at DSW are disregarded. Additionally, make the following adaptations: If no tile at UW , USE , SW , SE or DE is encountered, then project the whole column (which is a parallelogram of width one) in direction DSW . Otherwise, after performing the final tile shift in direction DE , repeatedly move S (on empty nodes) and enter the first tiled node at S or SE (which must exist since we did not project).

The following remarks aim to clarify the choices made in the above case distinction: In case C, the node $v + DE$ serves as a bridge node for v ; however, the agent takes an additional step by attempting to shift the tile to $v + SE + DSW$. This decision stems from the fact that $v + DE$ is not within the bounding cylinder of tiles observable from node v . Similarly, in case D, $v + DSW$ serves as a bridge node for v . Although $v + DSW$ is within the bounding cylinder of observable tiles, it shares the same x - and y -coordinates as v . It is essential to our analysis that tiles are never placed outside of the bounding cylinder, and that, except for projections, tiles consistently advance to the east or south. In case B, the agent removes the last tile of some column c and instead of immediately transitioning to the search phase, it first traverses the next column c' in the S direction. Similarly to the `BUILDPAR` procedure, where the agent first traverses the next column c' fully in direction N whenever a more northern column of the same x -coordinate as c is found, this additional traversal is crucial for the runtime analysis. To elaborate, if column c' has multiple adjacent columns to the west, then directly entering the search phase would result in repeatedly traversing the same tiles within column c' . However, with the additional traversal in direction S in case B (and in direction N in procedure `BUILDPAR`), we can ensure that each tile of c' is visited only a constant number of times whenever the agent does not currently perform a tile shift.

4 Analysis

Due to space constraints, the proofs of our lemmas are only sketched here. Complete proofs can be found in the full version of the paper [15]. In the following, $C^i = (\mathcal{T}^i, p^i)$ denotes the configuration that results from the execution of BUILDICICLE for i steps. We start by showing that our algorithm complies with the connectivity constraint of the 3D hybrid model.

► **Lemma 2.** *If the agent disconnects $G|_{\mathcal{T}^i}$ in step i , then $G|_{\mathcal{T}^{i+4}}$ is connected, and for all $i < j < i + 4$: $G|_{\mathcal{T}^j \cup \{p^j\}}$ is connected and the agent carries a tile.*

One can show that $G|_{\mathcal{T}^i}$ can only disconnect during the projection of a parallelogram of height one, and during consecutive tile shifts in cases D and E. These are the cases for which we explicitly preserve connectivity by moving not on, but instead adjacent to tiled neighbors.

► **Lemma 3.** *If during the execution of BUILDICICLE a tile is shifted from some node v to some node w , then there are tiled nodes $u_x, u_y \in \mathcal{T}$ with $x(w) = x(u_x)$ and $y(w) = y(u_y)$.*

The lemma is proven through an extensive case distinction that includes the four conditions under which a node is tiled during the BUILDPAR procedure, as well as any tile shift that may result when the agent exits the search phase at a node v that is not removable. There are at most $2^6 = 64$ such cases, since v can have tiled neighbors in at most six directions. We argue that the 20 cases depicted in Figure 5 are complete by providing 44 distinct neighborhoods for which v is removable. Similarly, we can prove the following lemma by considering cases where a tile is picked up instead of dropped.

► **Lemma 4.** *For each $i \geq 0$ there is a tiled node $v \in \mathcal{T}^i$ with $x(v) = 0$.*

We want to measure the progress of tiles within the bounding cylinder towards the east and south by considering their x - and y -coordinates. As part of the BUILDPAR procedure and cases B, D, and E, the y -coordinate of tiles can increase when their x -coordinate decreases. Although the size of the bounding cylinder cannot increase by Lemma 3, it may decrease. In such instances, by Lemma 4, the resulting bounding cylinder always aligns with the eastern side of the initial bounding cylinder $\mathfrak{C}(\mathcal{T}^0)$. To address this, we introduce a combined representation of the x - and y -coordinates w.r.t. the bounding cylinder $\mathfrak{C}(\mathcal{T})$ for arbitrary \mathcal{T} .

Let $y_{max}^{\mathcal{T}}$ and $y_{min}^{\mathcal{T}}$ be the maximum and minimum y -coordinates within $\mathfrak{C}(\mathcal{T})$, and let $h = y_{max}^{\mathcal{T}} - y_{min}^{\mathcal{T}} + 1$ be the *height* of $\mathfrak{C}(\mathcal{T})$, i.e., the cylinder's extent along the y -axis. We define the *xy-coordinate* of some node $v \in \mathfrak{C}(\mathcal{T})$ as $xy(v) = x(v) \cdot h + y(v) - y_{min}^{\mathcal{T}}$.

Consider the following definitions, which we refer to as P1–P3, that relate to some fragment $F \subseteq \mathcal{T}$. We show that at some point any configuration contains a fragment that fulfills P1–P3. Afterwards, we show that this configuration converges to an icicle.

► **Definition 5.** *Let $F \subset \mathcal{T}$ be an arbitrary fragment.*

- P1: F is a platform, if $\{v + X \mid v \in F, X \in \{UW, USE, UNE\}\} \cap \mathcal{T}$ is an empty set.
- P2: F is covering, if for each node $v \in \mathcal{T}$ there is a node $w \in F$ with $xy(w) = xy(v)$.
- P3: F is an aligned parallelogram, if for each node $v \in F$ it holds that for all i with $xy(v) \geq i \geq 0$ there is a node $w \in F$ with $xy(w) = i$.

P1 characterizes a locally uppermost fragment, P2 a fragment covering the xy -coordinates of all tiled nodes, and P3 a fragment wherein tiles have the shape of a parallelogram aligned along the southern, eastern, and northern sides of the bounding cylinder. We can now use P1–P3 to give an alternative definition of the icicle shape.

► **Definition 6.** A Configuration $C = (\mathcal{T}, p)$ is an icicle, if it contains a fragment F that satisfies P1–P3, and for any node $v \in \mathcal{T} \setminus F$ it holds that $v + \text{UNE} \in \mathcal{T}$.

Any tiled node that is not contained in the fragment F specified in Definition 6, must be somewhere $\text{DSW} = -\text{UNE}$ of F , as otherwise the number of tiles would be infinite. Hence, each node $v \in \mathcal{T} \setminus F$ is contained in a tower of tiles whose uppermost tile is contained in F , and thereby Definition 6 is equivalent to our definition of an icicle from Section 1.4.

Subsequently, we only consider configurations in which the agent leaves the search phase at some node v . This must eventually occur since moving upwards increases its z -coordinate, and moving in directions SW, NW, or N increases its xy -coordinate. Both coordinates are bounded within any finite set of tiled nodes. To simplify notation, we use $C^i = (\mathcal{T}^i, p^i)$ to represent the configuration where the agent leaves the search phase for the i -th time.

Consider the potential function $\Phi^i = \sum_{v \in \mathcal{T}^i} xy(v) + |\mathcal{P}^i|$, where \mathcal{P}^i denotes the set of all platforms, i.e., fragments satisfying P1. We first show its monotonicity and lower bound.

► **Lemma 7.** For each $i \geq 0$ it holds that $\Phi^i \geq \Phi^{i+1} \geq 0$, and if $\Phi^i = \Phi^{i+1}$, then (1) no tile was shifted between step i and $i + 1$, or (2) a fragment was projected between step i and $i + 1$.

The lemma mostly follows from the observation that within procedure BUILDPAR tiles are visited in decreasing order of their xy -coordinates, and that each tile shift in cases A–E decreases the x -coordinate of at least one tile. The number of platforms can only increase by one as a result of case D, which is compensated by two tiles with decreasing x -coordinate.

► **Lemma 8.** If $p^i \in F^i$ where F^i is a fragment in C^i that satisfies P1–P3, then $p^{i+1} \in F^{i+1}$ where F^{i+1} is a fragment in C^{i+1} that satisfies P1–P3.

The proof of the previous lemma is straight forward. Since F^i satisfies P1, the agent cannot leave F^i to a layer above. Since it satisfies P2 and cases A–E necessitate a tile below that is not covered by F^i , the agent must enter BUILDPAR within F^i . Finally, since F^i satisfies P3, the agent must project F^i directly after the BUILDPAR procedure. As a result, F^{i+1} is a direct copy of F^i in direction DSW which also satisfies P1–P3.

► **Lemma 9.** For each $i \geq 0$ there is a step $j > i$ such that (1) $\Phi^i > \Phi^j$ or (2) $p^j \in F^j$ where F^j is a fragment of configuration C^j that satisfies P1–P3.

If (2) holds in step i , the lemma follows from Lemma 8. Otherwise, by Lemma 7, either no tile was shifted or a projection was performed between step i and $i + 1$. In the first case, the agent must have progressed further west or upwards, which can happen only finitely many times since the configuration is finite. In the second case, we can show that after finitely many consecutive projections the agent is positioned in a fragment of larger size. Analogously the size of that fragment is bounded by the number of tiles, such that eventually either the potential decreases again or the latter statement (2) holds, concluding the lemma.

The initial number of platforms is at most n , and the initial xy -coordinate of any tiled node is at most n^2 . Hence, the initial potential is $\Phi^0 = \mathcal{O}(n^3)$. Consequently, Lemma 8 and Lemma 9 imply that eventually the agent is positioned in a fragment satisfying P1–P3.

For the second part of our analysis, dedicated to demonstrating convergence towards an icicle, we introduce another potential function Ψ^i . This function is defined as the number of empty nodes within the bounding box $\mathfrak{B}(\mathcal{T}^i)$ that have a tile somewhere in the DSW direction. Formally, $U^i = \{v \in \mathfrak{B}(\mathcal{T}^i) \setminus \mathcal{T}^i \mid v + k \cdot \text{DSW} \in \mathcal{T}^i \text{ for some } k > 0\}$ and $\Psi^i = |U^i|$.

► **Lemma 10.** Let $p^i \in F^i$ where F^i satisfies P1–P3. If $\Psi^i > 0$, then $\Psi^{i+1} < \Psi^i$.

The lemma is proven by showing that the projection of F^i results in at least one node in U^i to be tiled. Here, we can simply pick the node with maximum z -coordinate from U^i . Additionally, any node in U^{i+1} must already be contained in U^i , since the projection of F^i cannot create an empty node that has a tile somewhere UNE. Otherwise that tile would contradict that F^i is covering (P2) and a platform (P1).

Once the agent enters a configuration where it is positioned within a fragment satisfying P1–P3, it consistently remains within such a fragment in subsequent configurations according to Lemma 8. By Lemma 9, such a configuration must eventually be reached, and by the previous lemma, our second potential Ψ^i is strictly monotonically decreasing afterwards. It follows that the set of empty nodes within $\mathfrak{B}(\mathcal{T}^i)$ that have a tile somewhere in the DSW direction must eventually be empty. Consequently, any tiled node within the bounding box that is not contained in the singular uppermost fragment satisfying P1–P3 must possess a neighboring tile at UNE. Hence, the entire configuration satisfies Definition 6, which is captured by the following theorem, serving as the conclusion of our analysis:

► **Theorem 11.** *The sequence of configurations resulting from the execution of BUILDICICLE on any initially connected configuration $C^0 = (\mathcal{T}^0, p^0)$ with $p^0 \in \mathcal{T}^0$ converges to an icicle.*

5 Termination Criteria and Runtime

Once the agent is positioned within a fragment satisfying P1–P3, it remains within such a fragment and subsequently exclusively performs projections. In the case where the configuration is already an icicle (see Definition 6), every tiled node in the configuration must be traversed during these projections. This condition is essential for our termination check. The agent maintains a flag *term*, which it flags as true upon initiating a projection. This flag only reverts to false if the agent detects any violation of Definition 6 during the ongoing projection, i.e., whenever a tiled node v is observed for which $v + \text{UNE} \notin F$ and $v + \text{UNE} \notin \mathcal{T}$, where F is the fragment in which the projection was initiated. Once *term* still holds after a projection, the agent terminates. Note that the flag is reverted, even if $v + \text{UNE}$ is tiled immediately afterwards. As an example, if a tile shift from some node $w \in F$ to $v + \text{UNE}$ is performed as part of a projection, then node v is observed before the tile is placed at $v + \text{UNE}$. Although it is possible that the configuration is an icicle after tiling node $v + \text{UNE}$, the agent cannot verify it during that projection, as it does not traverse node v or any node DSW of v .

In general, by adhering to this termination procedure, the agent consistently performs one additional projection once the configuration converges to an icicle. Since the algorithm only terminates following a projection in which it could observe all tiled nodes and only if, in this case, Definition 6 is satisfied, the correctness of our algorithm is established.

The algorithm’s runtime can be expressed as the sum $t_{total} = t_{proj} + t_{shift} + t_{move}$, where t_{proj} accounts for all steps performed during the projection subroutine, t_{shift} for steps that are performed as part of some tile shift (outside of a projection), and t_{move} for any remaining step. We bound each term individually by $\mathcal{O}(n^3)$, which gives a runtime of $\mathcal{O}(n^3)$ in total.

► **Lemma 12.** *The total number of steps performed during projections is $t_{proj} = \mathcal{O}(n^3)$.*

The proof can be outlined as follows: Each projection takes time $\mathcal{O}(n)$. There are $\mathcal{O}(n)$ platforms initially, each of which require $\mathcal{O}(n)$ projections until the number $|\mathcal{P}|$ of platforms reduces by one. Additional platforms can only be created as a result of the execution of case D. For these platforms one can show that a single projection suffices to reduce the number of platforms. Additionally, the execution of case D decreases the x -coordinate of at least two tiles, which implies that at most $\frac{n^2}{2}$ platforms can be created.

► **Lemma 13.** *Let i be the first step following an arbitrary projection, and $j > i$ the next step in which a projection is initiated. Between step i and j at most $\mathcal{O}(n)$ steps are performed outside of tile shifts, and any tile shift from some node v to w takes $\mathcal{O}(xy(v) - xy(w))$ steps.*

The latter statement is easy to show. Especially, any tile shift in cases A–E requires only $\mathcal{O}(1)$ steps, but reduces the xy -coordinate of some tile by $\Omega(h)$. For the former statement, one must consider all cases in which the agent moves outside of tile shifts. The most challenging case is where the agent lifts the last tile of a column and then takes $\mathcal{O}(h)$ steps afterward. If that occurs in case B, then the above outlined difference between $\mathcal{O}(1)$ steps and a reduction of $\Omega(h)$ in the xy -coordinate accounts for the $\mathcal{O}(h)$ additional steps. If a column is removed in the BUILDPAR procedure, one can show that either the previous or the subsequent tile shift decreases the xy -coordinate of a tile by $\Omega(h)$, and the claim follows analogously.

As argued above, $\mathcal{O}(n^2)$ projections are performed in total, which together with Lemma 13 implies that $t_{other} = \mathcal{O}(n^3)$. The xy -coordinate of any tile is at most n^2 , non-increasing, and cannot be negative (see Lemmas 3 and 7). Together with Lemma 13, each tile contributes $\mathcal{O}(n^2)$ steps to t_{shift} , which implies that $t_{shift} = \mathcal{O}(n^3)$. This concludes our final theorem:

► **Theorem 14.** *BUILDICICLE has a runtime of $\mathcal{O}(n^3)$ steps.*

6 Experimental Analysis

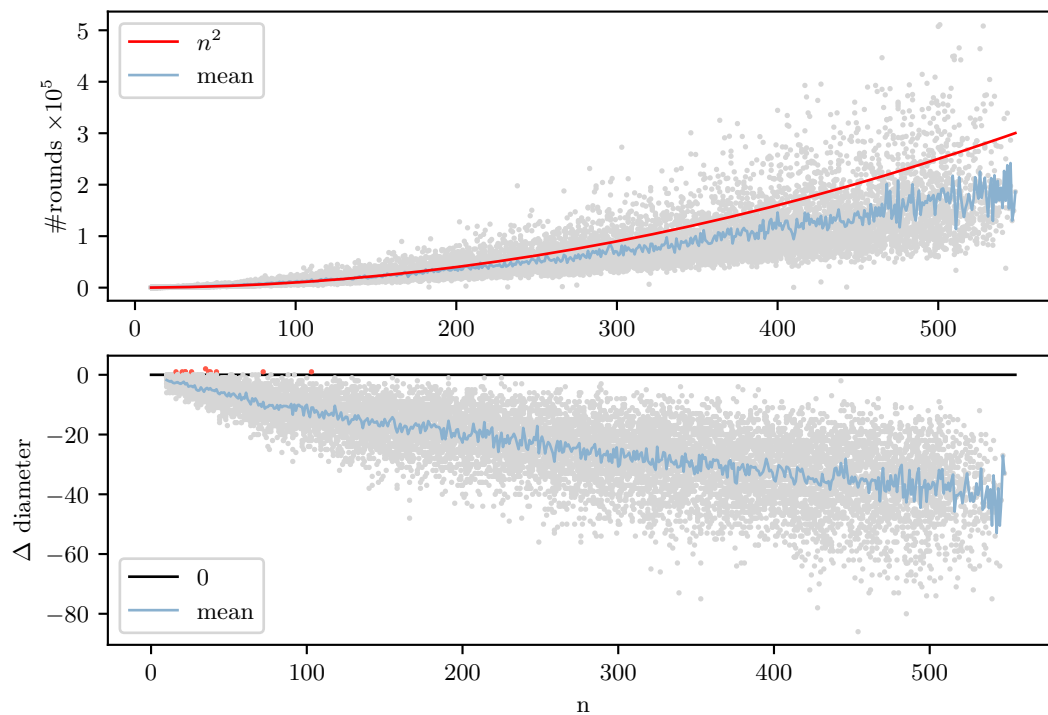
While our algorithm matches the runtime bound of the 3D line formation algorithm [16], the icicle offers distinct advantages over the line. The diameter of an icicle can be as low as $\mathcal{O}(n^{\frac{1}{3}})$, whereas a line consistently maintains a diameter of n . Unfortunately, our algorithm does not improve the diameter if the initial configuration already closely resembles a line. On the other hand, we conjecture that if the initial diameter is as low as $\mathcal{O}(n^{\frac{1}{3}})$ (which is the best case in 3D), then our algorithm can only increase the diameter by a factor of $\mathcal{O}(n^{\frac{1}{3}})$. We support our conjecture with the following simulation results on configurations where initially all tiles are contained in a sphere of radius $\mathcal{O}(n^{\frac{1}{3}})$.

We conducted a total of 12,250 simulations using the icicle formation algorithm on random configurations. For each value of n within the range $10 \leq n < 500$, we sampled 25 random configurations as follows: empty nodes were repeatedly chosen uniformly at random within a sphere of radius $4n^{\frac{1}{3}}$, and a tile was placed on each selected node until a connected component of tiled nodes with a size of at least n was formed. Subsequently, any tile outside of that component was removed, the agent was placed at a randomly chosen tile within the component, and the algorithm was simulated until termination. We measured the runtime as well as the difference in diameter, which are plotted in Figure 6.

Due to the nature of the described random generation process, configurations of size larger than 500 were also sampled, although less frequently. Specifically, we observed an average sampling rate of approx. 24.4 configurations for sizes at most 450, contrasting with approx. 15.4 configurations for sizes exceeding 450. This discrepancy contributes to the noticeable increase in variance as the configuration size approaches the 500 threshold.

The runtime remains well in the vicinity of n^2 , which can be attributed to the initial close packing of tiles in our random configurations. Instances where the diameter increases (highlighted by red dots) are infrequent, and their occurrence diminishes as the configuration size increases. This trend implies a general decrease in diameter in the average case.

We identified a configuration with an initial diameter of $\mathcal{O}(n^{\frac{1}{3}})$, where the diameter subsequently increases by a factor of $\Theta(n^{\frac{1}{3}})$. This particular configuration, which we consider to be the worst-case scenario, is discussed in Appendix B.



■ **Figure 6** The results stem from 12,250 simulations involving random configurations ranging in sizes from 10 to 550. The upper plot shows the number of steps until termination. The lower plot shows the difference in diameter between the input and output configurations. The simulations in which the diameter increases are highlighted by red dots.

7 Future Work

In this work, we introduced an algorithm capable of transforming any initially connected configuration into an icicle within $\mathcal{O}(n^3)$ steps, complemented by proofs of correctness and runtime analysis. While our algorithm’s experimental results are promising, future work should include a formal proof to substantiate the claimed upper bound of $\mathcal{O}(n^{\frac{1}{3}})$ on the increase in diameter. Additionally, the adaptability of our algorithm to the multi-agent case poses an intriguing challenge for future investigation. Given that the algorithm comprises distinct phases potentially executed in an interleaved manner, addressing its integration into a multi-agent framework presents a non-trivial research direction.

References

- 1 M. Akter, J. J. Keya, K. Kayano, A. M. R. Kabir, D. Inoue, H. Hess, K. Sada, A. Kuzuya, H. Asanuma, and A. Kakugo. Cooperative cargo transportation by a swarm of molecular machines. *Science Robotics*, 7(65), 2022. doi:10.1126/scirobotics.abm0677.
- 2 D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4), March 2006. doi:10.1007/s00446-005-0138-3.
- 3 M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, 1978. doi:10.1109/SFCS.1978.30.

- 4 J. Chao, J. Wang, F. Wang, X. Ouyang, E. Kopperger, H. Liu, Q. Li, J. Shi, J. hu, L. Wang, W. Huang, F. Simmel, and C. Fan. Solving mazes with single-molecule dna navigators. *Nature Materials*, 18, March 2019. doi:10.1038/s41563-018-0205-3.
- 5 H. Chen, C. Li, M. Mafarja, A. A. Heidari, Y. Chen, and Z. Cai. Slime mould algorithm: a comprehensive review of recent variants and applications. *International Journal of Systems Science*, 54(1), 2023. doi:10.1080/00207721.2022.2153635.
- 6 G.S. Chirikjian. Kinematics of a metamorphic robotic system. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994. doi:10.1109/ROBOT.1994.351256.
- 7 Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Amoebot - a new model for programmable matter. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, 2014. doi:10.1145/2612669.2612712.
- 8 Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication*, 2015. doi:10.1145/2800795.2800829.
- 9 Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal shape formation for programmable matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, 2016. doi:10.1145/2935764.2935784.
- 10 G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. Shape formation by programmable particles. *Distrib. Comput.*, 33(1), February 2020. doi:10.1007/s00446-019-00350-6.
- 11 P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. In *Mathematical Foundations of Computer Science 2004*, 2004.
- 12 R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, and C. Scheideler. Shape recognition by a finite automaton robot. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117, 2018. doi:10.4230/LIPIcs.MFCS.2018.52.
- 13 R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler, and T. Strothmann. Forming tile shapes with simple robots. *Natural Computing*, 19(2), June 2020. doi:10.1007/s11047-019-09774-2.
- 14 A. Heuer-Jungemann and T. Liedl. From dna tiles to functional dna materials. *Trends in Chemistry*, 1(9), 2019. doi:10.1016/j.trechm.2019.07.006.
- 15 K. Hinnenthal, D. Liedtke, and C. Scheideler. Efficient shape formation by 3d hybrid programmable matter: An algorithm for low diameter intermediate structures, 2024.
- 16 K. Hinnenthal, D. Rudolph, and C. Scheideler. Shape formation in a three-dimensional model for hybrid programmable matter. In *Proc. of the 36th European Workshop on Computational Geometry (EuroCG 2020)*, 2020.
- 17 F. Hoffmann. One pebble does not suffice to search plane labyrinths. In *International Conference on Fundamentals of Computation Theory*, 1981.
- 18 F. Hurtado, E. Molina, S. Ramaswami, and V. Sacristán. Distributed reconfiguration of 2d lattice-based modular robotic systems. *Autonomous Robots*, 38(4), April 2015. doi:10.1007/s10514-015-9421-8.
- 19 I. Kostitsyna, D. Liedtke, and C. Scheideler. Universal coating in the 3d hybrid model, 2023. doi:10.48550/arXiv.2303.16180.
- 20 I. Kostitsyna, C. Scheideler, and D. Warner. Fault-tolerant shape formation in the amoebot model. In *28th International Conference on DNA Computing and Molecular Programming (DNA 28)*, 2022. doi:10.4230/LIPIcs.DNA.28.9.
- 21 G. Li, D. St-Onge, C. Pinciroli, A. Gasparri, E. Garone, and G. Beltrame. Decentralized progressive shape formation with robot swarms. *Autonomous Robots*, 43(6), August 2019. doi:10.1007/s10514-018-9807-5.

- 22 H. Li, J. Gao, L. Cao, X. Xie, J. Fan, H. Wang, H. Wang, and Z. Nie. A dna molecular robot autonomously walking on the cell membrane to drive the cell motility. *Angewandte Chemie International Edition*, 60, September 2021. doi:10.1002/anie.202108210.
- 23 N. Nokhanji, P. Flocchini, and N. Santoro. Fully dynamic line maintenance by hybrid programmable matter. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2022. doi:10.1109/IPDPSW55747.2022.00087.
- 24 A. Padalkin, M. Kumar, and C. Scheideler. Shape formation and locomotion with joint movements in the amoebot model. *ArXiv*, abs/2305.06146, 2023.
- 25 M. J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2), June 2014. doi:10.1007/s11047-013-9379-4.
- 26 T. Peters, I. Kostitsyna, and B. Speckmann. Fast reconfiguration for programmable matter. In *37th International Symposium on Distributed Computing, DISC 2023*, 2023. doi:10.4230/LIPIcs.DISC.2023.27.
- 27 N. Tan, A. A. Hayat, M. R. Elara, and K. L. Wood. A framework for taxonomy and evaluation of self-reconfigurable robotic systems. *IEEE Access*, 8, 2020. doi:10.1109/ACCESS.2020.2965327.
- 28 P. Thalamy, B. Piranda, and J. Bourgeois. A survey of autonomous self-reconfiguration methods for robot-based programmable matter. *Robotics and Autonomous Systems*, 120, 2019. doi:10.1016/j.robot.2019.07.012.
- 29 A. J. Thubagere, W. Li, R. F. Johnson, Z. Chen, S. Doroudi, Y. L. Lee, G. Izatt, S. Wittman, N. Srinivas, D. Woods, E. Winfree, and L. Qian. A cargo-sorting dna robot. *Science*, 357(6356), 2017. doi:10.1126/science.aan6558.
- 30 T. Tucci, B. Piranda, and J. Bourgeois. A distributed self-assembly planning algorithm for modular robots. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018.
- 31 J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17(2), August 2004. doi:10.1007/s00446-003-0103-y.
- 32 H. Wang and M. Rubenstein. Shape formation in homogeneous swarms using local task swapping. *IEEE Transactions on Robotics*, 36(3), 2020. doi:10.1109/TR0.2020.2967656.
- 33 J. Werfel, K. Petersen, and R. Nagpal. Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172), 2014. doi:10.1126/science.1245842.
- 34 D. Woods, H. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, 2013. doi:10.1145/2422436.2422476.

A Deferred Pseudocode

The pseudocode for the 2D parallelogram formation algorithm, as detailed in Section 3.1, is given by Algorithm 1. Specifically, lines 12–34 within Algorithm 1 describe the BUILDPAR procedure, utilized by both the parallelogram and icicle formation algorithms. Note that the checks for tiles above (lines 13–14) can be disregarded in the 2D setting, as they only become relevant in the icicle formation algorithm. The PROJECT procedure is given in Algorithm 3, and the full icicle formation algorithm in Algorithm 2. Whenever multiple directions of movement are specified, their precedence is implicit in the provided order.

In Algorithm 1, the agent traverses a column in the S direction in lines 12–21 and the next column in the N direction in lines 26–31. Following the check for whether the empty node above the next column should be tiled (line 32), the agent recursively executes BUILDPAR starting at the N-most node of the next column (line 33). The procedure may return with the agent being in various states, such as positioned on a tiled or empty node, with or without a tile. In the main loop of the algorithm, BUILDPAR is executed repeatedly, and distinctions between these states are made to either terminate (line 4), retrieve the tile at which BUILDPAR was previously entered (lines 5–11), or enter the search phase (line 2).

In Algorithm 2, lines 16–22 are dedicated to handling case B, where a tile is shifted in the DE direction to maintain connectivity. Lines 23–29 cover case D, and lines 30–40 cover case E. These cases involve shifting multiple tiles of a column in the DE direction. For concise pseudocode, the handling of case A is delegated to the BUILDPAR procedure, and the check for a tile at $v + SE + DSW$ from case C is integrated into lines 4–5.

B Worst-Case Configuration

In the following, we present what we believe to be the worst-case configuration. Consider the configuration C depicted in Figure 7a that consists of three layers. The middle layer contains $k = \Theta(n^{\frac{1}{3}})$ fragments F_1, \dots, F_k ordered from east to west, where each F_i has size

■ **Algorithm 1** 2DPARALLELOGRAMFORMATION.

```

1 while true do
2   while  $\{p + NW, p + SW, p + N\} \cap \mathcal{T} \neq \emptyset$  do move to tile at NW, SW or N
3   firstColumn  $\leftarrow$  true; run BUILDPAR
4   if  $p \notin \mathcal{T}$  then return  $\triangleright$  terminate S of easternmost column
5   else if  $r$  carries no tile then
6     if firstColumn then
7       while  $p + N \in \mathcal{T}$  do move N
8     else
9       while  $\{p + SW, p + S\} \cap \mathcal{T} \neq \emptyset$  do move to tile at SW or S
10      while  $\{p + NW, p + SW, p + N\} \cap \mathcal{T} \neq \emptyset$  do move to tile at NW, SW or N
11      pickup tile; move to tile at S, SE or NE

  procedure BUILDPAR
12 while  $p \in T$  do
13   if  $p + UW \in \mathcal{T}$  or  $p + USE \in \mathcal{T}$  or  $p + UNE \in \mathcal{T}$  then  $\triangleright$  irrelevant in 2D
14     move to tile at UW, USE or UNE ; return
15   else if firstColumn and  $p + SW \in \mathcal{T}$  then
16     move SW; return  $\triangleright$  found more western column
17   else if  $p + NE \in \mathcal{T}$  and  $p + SE \notin \mathcal{T}$  then
18     move SE; place tile; move NW; return  $\triangleright$  place tile below eastern column
19   else if  $p + N, p + SE \in \mathcal{T}$  and  $p + NE \notin \mathcal{T}$  then
20     move NE; place tile; move SW; return  $\triangleright$  place tile above eastern column
21   move S
22 if  $p + N, p + NE, p + SE \in \mathcal{T}$  then
23   place tile; move N  $\triangleright$  place tile below current column
24 else if  $p + NE \in \mathcal{T}$  then
25   move NE; move N; firstColumn  $\leftarrow$  false  $\triangleright$  move to top of next column
26   while  $p \in \mathcal{T}$  do
27     if  $p + SW \notin \mathcal{T}$  and  $p + NW \in \mathcal{T}$  then
28       while  $p + N \in \mathcal{T}$  do move N
29       while  $p + NW \notin \mathcal{T}$  do move S
30     return  $\triangleright$  found more northern column
31   move N
32 if  $p + S, p + SE \in \mathcal{T}$  then place tile  $\triangleright$  place tile above current column
33 else move S; run BUILDPAR
34 return

```

Algorithm 2 BUILDICICLE.

```

1 while true do
2   while  $\{p + X \mid X \in \{UW, USE, UNE, NW, SW, N\}\} \cap T \neq \emptyset$  do
3     | move to tile at UW, USE, UNE, NW, SW or N
4   if  $G|_{N_{\mathcal{T}}(p)}$  or  $G|_{N_{\mathcal{T}}(p) \cup \{p+SE\}}$  is connected or  $G|_{N_{\mathcal{T}}(p) \cup \{p+SE+DSW\}}$  is connected
   with  $p + SE + DSW \in \mathcal{T}$  then
5     |  $firstColumn \leftarrow true$ ; run BUILDPAR
6     | if  $p \notin \mathcal{T}$  then move N; run PROJECT
7     | else if  $r$  carries no tile then ...  $\triangleright$  same as lines 8–15 from Algorithm 1
16  else if  $G|_{N_{\mathcal{T}}(p) \cup \{p+DE\}}$  is connected then
17    | if  $N_{\mathcal{T}}(p) = \{p + DSW, p + SE\}$  then
18      | move SE + DSW; place tile; move UNE + NW; pickup tile; move SE
19    | else move DE; place tile; move UW; pickup tile
20    | if  $p + SE, p + NE \in \mathcal{T}$  and  $p + S \notin \mathcal{T}$  then
21      | move SE; while  $\{p + UW, p + USE, p + SW, p + S\} \cap \mathcal{T} = \{p + S\}$  do move S
22    | else move to tile at S, SE or NE
23  else if  $N_{\mathcal{T}}(p) = \{p + DNW, p + S, p + NE\}$  then
24    | while  $\{p + X \mid X \in \{UW, USE, SW, DSW, SE, DE, S\}\} \cap \mathcal{T} = \{p + S\}$  do move S
25    | if  $\{p + X \mid X \in \{UW, USE, SW\}\} \cap \mathcal{T} = \emptyset$  then
26      | if  $p + DE \in \mathcal{T}$  then move N
27      | move DE; place tile; move UW; pickup tile
28      | while  $p + N \in \mathcal{T}$  do move SE + DNW; place tile; move UW; pickup tile
29      | move NE
30  else if  $N_{\mathcal{T}}(p) = \{p + DNW, p + S\}$  then
31    | while  $\{p + X \mid X \in \{UW, USE, SW, SE, DE, S\}\} \cap \mathcal{T} = \{p + S\}$  do move S
32    | if  $\{p + X \mid X \in \{UW, USE, SW\}\} \cap \mathcal{T} = \emptyset$  then
33      | if  $\{p + X \mid X \in \{SE, DE\}\} \cap \mathcal{T} = \emptyset$  then move N; run PROJECT
34      | else
35        | ...  $\triangleright$  same as lines 26–28
38      | while  $p \notin \mathcal{T}$  do
39        | | move S; if  $p + SE \in \mathcal{T}$  then move SE

```

$\mathcal{O}(i)$ and the agent's initial position is $p^0 \in F_1$. Additionally, the configuration contains a fragment F_0 of size one east of the agent's initial position. Observe that the bounding box of F_i contains no node from F_{i+1} for any i with $0 < i < k$. It follows that the agent builds and projects parallelograms in the order F_1, \dots, F_k . Since the bounding box of F_i contains p^0 for all $i > 0$, it further follows that k tiles are projected from p^0 in direction DSW. Only then, the agent traverses the lower layer and eventually finds fragment F_0 where it moves further upwards. Now consider the configuration that consists of $\Theta(n^{\frac{1}{3}})$ copies of C in direction UNE (see Figure 7b). That configuration has diameter $\mathcal{O}(n^{\frac{1}{3}})$ initially. Throughout the icicle formation algorithm, some tile at node p^0 is projected $\Theta(n^{\frac{2}{3}})$ times, which implies that the resulting icicle has depth and thereby also diameter $\Theta(n^{\frac{2}{3}})$.

Algorithm 3 PROJECT.

```

procedure PROJECT
1 if  $p + N, p + S \notin \mathcal{T}$  then                                     ▷ parallelogram of height one
2   do
3     while  $p \in \mathcal{T}$  do move DSW
4     place tile; while  $p + UNE \in \mathcal{T}$  do move UNE
5     pickup tile
6     if  $p + NW \in \mathcal{T}$  then move SW; move DNW else move DSW; return
7   while  $p + UNE \in \mathcal{T}$ 
8 else
9   do
10    while  $p + N \in \mathcal{T}$  do move N
11    while  $p \in \mathcal{T}$  do move DSW
12    place tile; while  $p + UNE \in \mathcal{T}$  do move UNE
13    pickup tile
14    if  $p + S \in \mathcal{T}$  then move S
15    else if  $p + NW \in \mathcal{T}$  then move NW
16    else move DSW; return
17  while  $p \in \mathcal{T}$ 
    
```

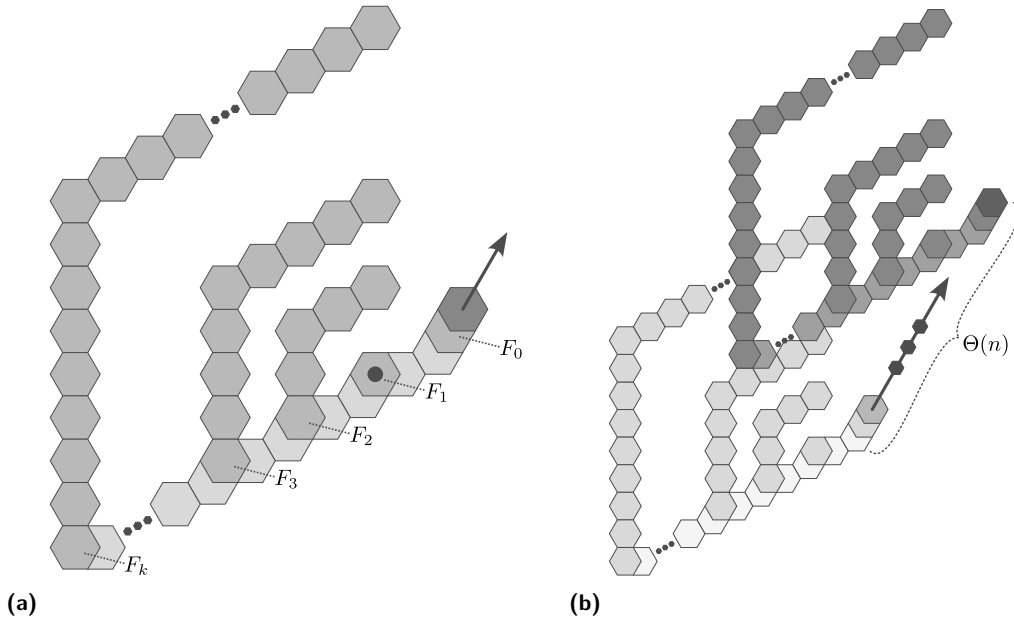


Figure 7 Illustrating what we believe to be the worst case configuration in terms of increase in diameter. In (a), the three lowest layers of the configuration are depicted in detail. The second-lowest layer contains $k = \Theta(n^{1/3})$ fragments F_i , each of size $\mathcal{O}(i)$, and the agent's initial position $p^0 \in F_1$. Observe that the bounding box of each F_i contains p^0 . The whole configuration is depicted in (b) and consists of $\Theta(n^{1/3})$ copies of the layers depicted in (a) in direction UNE (indicated by the arrows).