# Sublinear-Time Reconfiguration of Programmable Matter with Joint Movements

**Manish Kumar**
Indian Institute of Technology Ropar, India

**Othon Michail**
University of Patras, Greece

**Andreas Padalkin**
Paderborn University, Germany

**Christian Scheideler**
Paderborn University, Germany

## Abstract

We study centralized reconfiguration problems for geometric amoebot structures. A set of $n$ amoebots occupy nodes on the triangular grid and can reconfigure via expansion and contraction operations. We focus on the joint movement extension, where amoebots may expand and contract in parallel, enabling coordinated motion of larger substructures. Prior work introduced this extension and analyzed reconfiguration under additional assumptions such as metamodules, i.e., collections of modules that act as a single unit.

In contrast, we investigate the intrinsic dynamics of reconfiguration without such assumptions by restricting attention to centralized algorithms, leaving distributed solutions for future work. We study the *reconfiguration problem* between two classes of amoebot structures $A$ and $B$: For every structure $S \in A$, the goal is to compute a schedule that reconfigures $S$ into some structure $S' \in B$. Our focus is on sublinear-time algorithms.

We affirmatively answer the open problem by Padalkin *et al.* (Auton. Robots, 2025) whether a within-the-model sublinear-time universal reconfiguration algorithm is possible, by proving that any structure can be reconfigured into a *canonical* line-segment structure in $O(\sqrt{n}\log n)$ rounds. Additionally, we give a constant-time algorithm for reconfiguring any spiral structure into a line segment. These results are enabled by new constant-time primitives that facilitate efficient parallel movement. Our findings demonstrate that the joint movement model supports sublinear reconfiguration without auxiliary assumptions. A central open question is whether universal reconfiguration within this model can be achieved in polylogarithmic or even constant time.

**2012 ACM Subject Classification** Computing methodologies → Cooperation and coordination; Theory of computation → Computational geometry

**Keywords and phrases** amoebot model, programmable matter, modular robot system, reconfiguration

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2026.11

**Related Version** *Full Version*: `https://arxiv.org/abs/2603.10720` [14]

**Funding** *Andreas Padalkin* and *Christian Scheideler*: These authors were supported by the DFG Project SCHE 1592/10-1.

**Acknowledgements** We thank Christian Bauer and Leo Decking for the software implementation of the proposed algorithms.

## 1 Introduction

Programmable matter is made of many small, identical robotic modules that can change the properties of matter, such as its shape, color, or density, in a programmable way [19]. In this paper, we consider the reconfiguration dynamics of the geometric amoebot model [7, 8, 9]





from a centralized perspective. In the distributed version of this model, a set of $n$ robots (called *amoebots*) is placed on a triangular grid. Each amoebot can communicate with its neighboring amoebots and move by expanding into an empty adjacent node and then contracting into it. Since movement and communication happen locally from node to node, many problems in this model have a natural lower bound of $\Omega(D)$ rounds, where $D$ is the diameter of the structure.

To overcome this limitation, previous work introduced the *joint movement extension* [15]. This extension allows multiple amoebots to move together in a coordinated way, e.g., an amoebot can push or pull other amoebots while expanding or contracting, which enables large parts of the structure to move in parallel. This raises an important question: Can we reconfigure an amoebot structure much faster using joint movements? By "faster" we mean running in sublinear parallel time (measured in discrete rounds), or, if possible, even in polylogarithmic or constant time. In this paper, we are interested in whether this is in principle feasible, thus we naturally focus on the centralized reconfiguration dynamics of the model.

Actuation operations that can globally affect the structure are sometimes called *linear-strength operations*, representing the property that they have sufficient strength to displace up to a linear-sized set. Although such operations may be a strict assumption to make in some systems, they have been incorporated in a variety of theoretical models [2, 3, 4, 5, 13, 20] and are reasonable in several real-world contexts like micro- and nano-robotic systems in low-viscosity environments, DNA nanotechnology, and programmable matter where a single local operation can directly cause reconfiguration of the entire system. They also form a simple theoretical abstraction for dynamics that, in practice, could be implemented by joint movements, with multiple robots jointly bearing the load of a linear-strength operation. We use the terms *move* and *movement* to refer to actuation operations throughout the paper.

Aloupis *et al.* [4, 5] studied a model of a modular robot system known as *crystalline robots* [18]. The 2D version of the crystalline model represents modules as squares on a 2D grid, forming a connected shape of modules attached to adjacent modules. Each individual module can expand and contract, by extending one of its faces one unit out and retracting it back at some later point. Due to modules being attached to each other, up to linear-size components can move due to a module's expansion or contraction. In [5], they gave a universal centralized reconfiguration algorithm for the crystalline model that, for any pair of connected shapes $S_I, S_F$ of the same number of modules $n$, can transform $S_I$ into $S_F$ in $O(\log n)$ rounds. Note that in the crystalline model reconfiguration of some classes of shapes is impossible without metamodules (or other assumptions), and these algorithms essentially rely on the use of metamodules.

Recent work has showed that joint movements in the geometric amoebot model can also be very powerful when additional assumptions such as metamodules are used [15]. However, it is still unclear how powerful the model is on its own, without relying on extra assumptions. In this paper, we study the centralized reconfiguration problem in the geometric amoebot model with joint movements, without relying on such additional assumptions, thus focusing on the limits of the model dynamics themselves.

We consider centralized algorithms, where a global scheduler knows the full structure and decides the movements of all amoebots in each round. This allows us to focus on the main question: How fast can reconfiguration be in principle? Given two classes of amoebot structures, the goal is to design sequences of movements that reconfigure every structure from one class into some structure of the other class. An algorithm in this context is a formal description of these sequences of movements for all structures in the source class.

Our main result is the first within-the-model universal sublinear-time reconfiguration algorithm. We show that any connected amoebot structure can be reconfigured into a line segment in $O(\sqrt{n}\log n)$ rounds. Since any two connected structures with the same number of amoebots can both be reconfigured into a line segment, and every reconfiguration is reversible, this immediately implies that we can reconfigure any structure into any other structure within the same asymptotic bound. Therefore, we obtain universal reconfiguration in $O(\sqrt{n}\log n)$ rounds. This answers an open question on whether large-scale reconfiguration can be achieved in sublinear time without additional assumptions [15]. We also present an algorithm that reconfigures spiral structures into a line segment in constant time. This result shows that non-trivial subclasses of paths can be reconfigured extremely fast within the model. We hope that this can serve as a foundational step toward polylogarithmic or even constant-time universal reconfiguration, which is left as a central open question.

Overall, our results show that joint movements can greatly increase the *reconfiguration efficiency* of programmable matter structures, i.e., how quickly a structure can be reconfigured into another. While the original amoebot model is limited by diameter-based lower bounds, as we show, coordinated parallel motion enables significantly faster global reconfiguration.

**Organization of the paper.** In Section 2, we describe the geometric amoebot model with joint movements and formally define the reconfiguration problem and the classes of structures considered in this paper. Section 3 introduces a set of basic movement primitives that will be used as building blocks in our algorithms. In Section 4, we present an algorithm that reconfigures monotone structures into a line segment, which will become an important subroutine in our main algorithms. Section 5 describes our main result: a universal reconfiguration algorithm that reconfigures any connected structure into a line segment in sublinear time. Section 6 presents algorithms for reconfiguring spiral structures into a line segment, including a constant-time solution. Finally, Section 7 discusses open problems.

## 2 Model and Problem Statement

In this section, we formally present the model and the problem statement. In Section 2.1, we present the geometric amoebot model with joint movements. In Section 2.2, we define the reconfiguration problem and the classes of amoebot structures considered in this paper.

### 2.1 Geometric Amoebot Model with Joint Movements

In this section, we present the geometric amoebot model [8, 9] with the joint movement extension [15]. The amoebot structure consists of $n$ amoebots placed on the infinite regular triangular grid graph $G_\Delta = (V_\Delta, E_\Delta)$ (see the left side of Figure 1). Each amoebot either occupies a single node or two adjacent nodes and the edge between them. We call an amoebot *contracted* if the amoebot occupies a single node and *expanded* otherwise. Every node of $G_\Delta$ is occupied by at most one amoebot. Adjacent amoebots are connected by *bonds*.

Let $C$ denote the set of nodes occupied by contracted amoebots, $X$ the set of nodes occupied by expanded amoebots, $A$ denote the set of all edges occupied by expanded amoebots, and $B$ the set of all bonds. We define the *connectivity graph* of an amoebot structure as the graph $S = (C \cup X, A \cup B)$. The edges of $A$ are pairwise disjoint. We assume that initially, $S$ is connected. To each edge in $A \cup B$, we assign a geometric orientation. We only allow orientations parallel to the axes of the triangular grid $G_\Delta$ (see the right side of Figure 1). The *amoebot structure* is defined by its connectivity graph and the assignment of orientations. For the sake of simplicity, we will denote each amoebot structure by its connectivity graph.

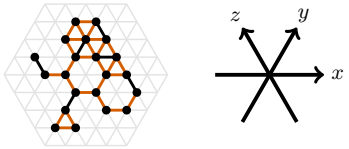
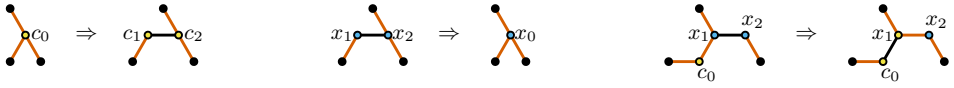



**Figure 1** Amoebot model and axes. The black nodes and edges indicate the nodes and edges occupied by amoebots. The red edges indicate bonds between the amoebots. The gray edges indicate the triangular grid $G_\Delta$. We omit the grid in all other figures.




**(a)** Expansion. **(b)** Contraction. **(c)** Handover.

**Figure 2** Movements. The yellow amoebot in (a) and (c) indicates amoebot $c$. The blue amoebot in (b) and (c) indicates amoebot $x$.

We assume the *fully synchronous* activation model, i.e., time is divided into synchronous rounds, and every amoebot is active in each round. In each round, each amoebot can perform a single movement which is performed in two steps. Let $S_i = (C_i \cup X_i, A_i \cup B_i)$ be the amoebot structure at the beginning of the round.

In the **first step**, the amoebots remove bonds from $S_i$ as follows. Each amoebot can decide to release an arbitrary subset of its currently incident bonds in $S_i$. A bond is removed if and only if one of the incident amoebots releases the bond. Note that we only remove bonds (i.e., edges in $B_i$) but no occupied edges (i.e., edges in $A_i$). Let $R_i \subseteq B_i$ be the set of the remaining bonds and $S'_i = (C_i \cup X_i, A_i \cup R_i)$ the resulting amoebot structure. We require that $S'_i$ is connected since otherwise, disconnected parts might float apart. We say that a *connectivity conflict* occurs if and only if $S'_i$ is not connected. Whenever a connectivity conflict occurs, the amoebot structure transitions into an undefined state such that we become unable to make any statements about the structure.

In the **second step**, each amoebot may perform one of the following movements. A contracted amoebot $c$ occupying $c_0 \in C_i$ may *expand* on one of the axes of the grid (see Figure 2a). For that, we replace $c_0$ by two nodes $c_1, c_2$ and an edge $a = \{c_1, c_2\}$, and assign an orientation to $a$. For each to $c$ incident bond $b \in R_i$, we replace $c_0$ with one of the new nodes. Note that the incident bonds do not change their orientations. As a result, all connected amoebots move with the expanding amoebot. An expanded amoebot may *contract* analogously by reversing the contraction (see Figure 2b).

Furthermore, pairs of amoebots may perform isolated *handovers* as follows (see Figure 2c). Consider a contracted amoebot $c$ occupying $c_0 \in C_i$ and an expanded amoebot $x$ occupying $x_1, x_2 \in X_i$ and $a = \{x_1, x_2\} \in A_i$ that are connected by a bond $b = \{c_0, x_1\} \in R_i$. Intuitively, we want to switch the association of the expanded amoebot and the bond. More precisely, (i) amoebot $c$ becomes expanded and occupies nodes $c_0$, $x_1$, and edge $b$, (ii) amoebot $x$ becomes contracted and occupies node $x_2$, and (iii) edge $a$ becomes a bond.

At the end of the movements, let $C_{i+1}$ denote the set of nodes occupied by contracted amoebots, and $X_{i+1}$ the set of nodes occupied by expanded amoebots, $A_{i+1}$ the set of all edges occupied by expanded amoebots, and $R'_i$ the set of all bonds. Let $S''_i = (C_{i+1} \cup X_{i+1}, A_{i+1} \cup R'_i)$ be the resulting connectivity graph. We require that $S''_i$ is a subgraph of $G_\Delta$ in compliance with the orientations of all edges (i.e., $A_{i+1} \cup R'_i$). We say that a *collision* occurs if and only if $S''_i$ is not a subgraph of $G_\Delta$, i.e., either the amoebots cannot be mapped to the triangular

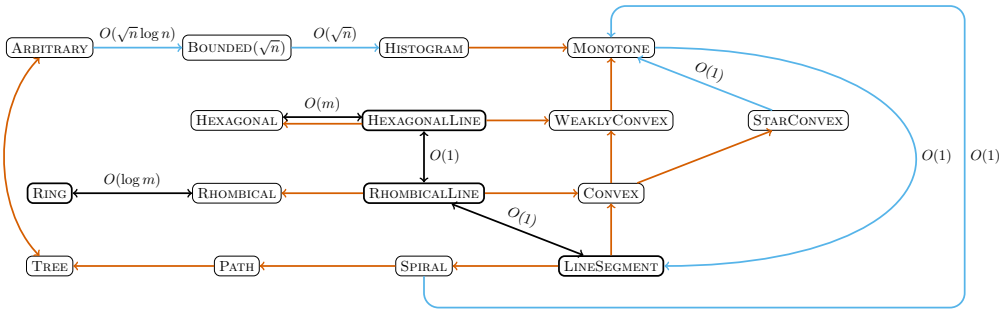



**Figure 3** Results. Classes with a thicker frame indicate canonical classes. Red arrows point to superclasses and with that indicate trivial solutions. Note that we do not depict all superclasses. Black arrows indicate results by Padalkin *et al.* [15]. Note that their superconstant algorithms simulate algorithms from [5, 12]. They define rhombical and hexagonal metamodules. Rhombical (Hexagonal) contains all amoebot structure that consist of rhombical (hexagonal) metamodules. RhombicalLine (HexagonalLine) contains all amoebot structure forming a line of rhombical (hexagonal) metamodules. Ring contains canonical amoebot structures forming a ring (see [5] for details). $m$ denotes the number of metamodules. Blue edges indicate our results. Note that our algorithm that reconfigures bounded structures into histograms simulates an algorithm from [4].

grid or multiple amoebots are mapped to the same node. Whenever a collision occurs, the amoebot structure transitions into an undefined state such that we become unable to make any statements about the structure. We, as algorithm designers, are responsible to make sure collisions do not happen. In the next round, we continue with the *adjacency closure* $S_{i+1} = (C_{i+1} \cup X_{i+1}, A_{i+1} \cup B_{i+1})$ of $S_i''$, i.e., we add bonds between amoebots until all adjacent amoebots are connected.

In this paper, we assume that we have a centralized scheduler. The scheduler knows the current state of the amoebot structure at all times. At the beginning of each synchronous round, it decides for each amoebot which bonds to release, and which movement to perform. Due to our focus being on understanding the feasibility of sublinear-time dynamics within the model (i.e., no additional assumptions in contrast to previous work which used metamodules), we leave the design of distributed solutions for future work.

## 2.2 Problem Statement and Classes of Amoebot Structures

In this paper, we consider the *reconfiguration problem* between two classes $A$ and $B$ of amoebot structures of the same size. The goal of the problem is to develop an algorithm that for every amoebot structure $S \in A$ computes a schedule that reconfigures $S$ into any amoebot structure $S' \in B$. Note that an algorithm may map the amoebot structures of $A$ to only a subset of $B$ and not the whole class. Hence, in general, it is not possible to use the same algorithm with reversed schedules to solve the reverse problem. In the remainder of this paper, we will consider the following classes (see Figure 3, which also shows our results). We assume that all structures in the classes only consist of contracted amoebots. Note that each class is closed under translation, rotation, and reflection.

Arbitrary: All structures.

Tree: Structures forming a tree. Note that this class is equal to Arbitrary since each structure has a spanning tree.

Path: Structures forming a path.

Spiral: Structures forming a simple spiral with 120° corners. Note that a single line segment is a one-segmented spiral.

LineSegment: Structures that form a line segment.

Monotone: We call the intersection of the amoebot structure with a line parallel to the $x$-axis an *$x$-section*. Note that an $x$-section is not necessarily connected. We define *$y$- and $z$-sections* analogously. In particular, we also call $x$-sections *rows*, and $y$-sections *columns*. We call an amoebot structure *$x$-monotone* if and only each $x$-section is connected. We define $y$- and $z$-monotone analogously. We call an amoebot structure *monotone* if and only if it is $x$-monotone, $y$-monotone, or $z$-monotone.

Histogram: We call a line segment of amoebots parallel to the $x$-axis an *$x$-segment*. We define *$y$- and $z$-segments* analogously. We say that an amoebot structure forms a *histogram* if and only if it consists of an $x$-segment with $y$-segments attached to one side of the $x$-segment, or can be obtained from such a structure by reflections and rotations. Note that every histogram is monotone.

Bounded($k$): Structures with at most $k$ non-empty $x$-, $y$-, or $z$-sections.

Convex: We call an amoebot structure *convex* if and only if for each pair of amoebots $u, v$, all shortest paths between $u$ and $v$ are part of the amoebot structure.

WeaklyConvex: We call an amoebot structure *weakly convex* if and only if for each pair of amoebots $u, v$, at least one shortest path between $u$ and $v$ is part of the amoebot structure.

StarConvex: We call an amoebot structure *star-convex* if and only if there is an amoebot $u$ such that for each amoebot $v$, all shortest paths between $u$ and $v$ are part of the amoebot structure [6].

Another interesting variant of the *reconfiguration problem* is the one within a class $A$. The goal of this variant is to develop an algorithm that for any two amoebot structures $S, S' \in A$ (with the same number of amoebots) computes a schedule that reconfigures $S$ into $S'$. We obtain a universal reconfiguration algorithm if we solve the problem for the class Arbitrary. A common way to design such an algorithm is to use a canonical structure $C$ as an intermediate structure during the reconfiguration (e.g., [4, 5, 12]). Given schedules from $S$ and $S'$ to $C$, we can obtain a schedule from $S$ to $S'$ by first executing the schedule from $S$ to $C$ and then the reverse schedule from $S'$ to $C$. In this paper, we utilize a line segment as the canonical structure. Our focus therefore is to solve the reconfiguration problem between various classes and the class LineSegment. Our main results are a universal reconfiguration algorithm that requires $O(\sqrt{n}\log n)$ rounds and an algorithm that reconfigures a spiral into a line in $O(1)$ rounds. Further results can be found in Figure 3.

## 3 Preliminaries

In this section, we present basic movement primitives: the tunneling, shearing, parallelogram, triangle, and trapezoid primitive. These are core components in our reconfiguration algorithms. Due to space constraints, we omit the description of the implementation of these primitives and refer to the full version.

In the tunneling primitive, originally defined in [10], we move a chain of amoebots along a path by constantly performing handovers.[1] We will use this primitive to "tunnel" an amoebot from one end to the other end without changing the structure of the remaining chain. In

[1] In previous work, the primitive is defined for spanning forests and is therefore called the spanning forest primitive. Since we only need the simpler case of a path, we refer to [7, 10] for the general case.

**Figure 4** Tunneling primitive. In each round, adjacent amoebots of the same color perform a handover. The gray node is unoccupied. In all figures, a simple arrow between two subfigures indicates the execution of the first step of a round (i.e., the removal of bonds) after establishing the adjacency closure, and a double arrow the execution of the second step of a round (i.e., the movements) unless stated otherwise. Note that only the fully completed rounds (i.e., both steps were executed) count to the total number of rounds.

general, it may take up to $O(n)$ rounds to tunnel the amoebot through the chain. We call a chain that alternates between contracted and expanded amoebots *alternating*. On alternating chains, the primitive only requires 3 rounds (see Figure 4). Note that the primitive actually moves all amoebots along the path without changing the order. However, since the amoebots are identical, the outcome is equivalent to a real tunneling. By construction, we obtain the following lemma.

▶ **Lemma 1.** *On alternating chains, the tunneling primitive requires* 3 *rounds to tunnel an amoebot through the chain.*

We perform all remaining primitives on substructures of the amoebot structure. Other parts of the amoebot structure may be connected to that substructure at specific amoebots which we call *connection points*. Each primitive has two of these connection points. As the parts are not necessarily connected without the substructure, we must keep it connected at all times. With the exception of the shearing primitive, we maintain the relative position between those connection points.

In the shearing primitive, we shear a line segment of amoebots to another axis.[2] We emphasize that this is not a rotation of the line segment since the attached parts of the amoebot structure are not rotated with the line segment. The endpoints of the line serve as the connection points.

▶ **Lemma 2.** *The shearing primitive requires* 2 *rounds.*

Consider a parallelogram where only the two sides incident to an obtuse corner are occupied by amoebots. The amoebots at the acute corners serve as the connection points. The goal of the parallelogram primitive is to move the amoebots to the other two sides without leaving the parallelogram and without changing the relative positions of the connection points at any time. Note that these sides require the same number of amoebots as the initially occupied sides.

▶ **Lemma 3.** *The parallelogram primitive requires* 2 *rounds.*

Consider an equilateral triangle where only two sides (w.l.o.g., the legs) are occupied by amoebots. The amoebots at the corners incident to the unoccupied side (i.e., the base) serve as the connection points. The goal of the triangle primitive is to occupy the base and one of the legs without leaving the triangle and without changing the relative positions of the connection points at any time. Note that any two sides of the triangle require the same number of amoebots since the triangle is equilateral.

[2] This primitive is similar to the realignment primitive of [15]. In fact, both primitives use the same intermediate structure. This allows us to switch between all structures of both primitives.

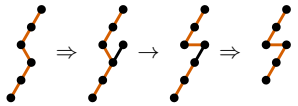


**Figure 5** Preprocessing phase of the Monotone2LineSegment algorithm.

▶ **Lemma 4.** *The triangle primitive requires $O(1)$ rounds.*

Finally, consider a trapezoid where the shorter base and the legs are occupied by amoebots. The amoebots at the acute corners serve as the connection points. Let a node on the longer base be given. We will call it the *starting point*. The goal of the trapezoid primitive is to occupy the longer base and a path between the bases starting from the starting point, which may consist of up to 2 segments, without leaving the trapezoid and without changing the relative positions of the connection points at any time. Note that the shorter base and both legs require the same number of amoebots as the longer base and one leg. Furthermore, the path between the legs requires the same number of amoebots as one leg. Hence, we have the exact number of amoebots for this reconfiguration.

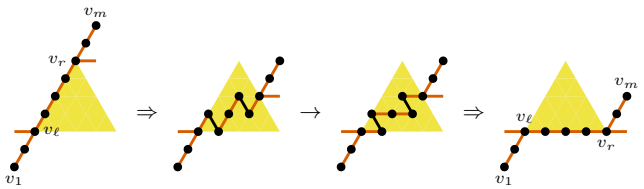
▶ **Lemma 5.** *The trapezoid primitive requires $O(1)$ rounds.*

## 4 Monotone Structures

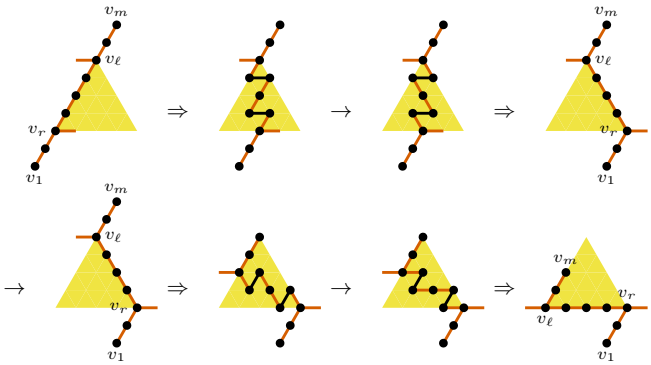
Both our universal reconfiguration algorithm and our constant-time Spiral2Line algorithm make use of monotone structures as intermediate structures. In the following, we present our Monotone2LineSegment algorithm. W.l.o.g., we assume that the amoebot structure is $y$-monotone. By definition, each column is a $y$-segment. We first remove all bonds between adjacent columns except for one arbitrary horizontal bond. If there is no such bond, we shift the right column up by one position in a preprocessing phase (see Figure 5).

Let $(v_1, \dots, v_m)$ denote the amoebots of the column from bottom to top. Let $v_\ell$ ($v_r$) be the amoebot connected to the left (right) column if it exists and $v_\ell = v_1$ ($v_r = v_m$) otherwise. Each column applies the following rules until $m = 1$ (see Figure 6).

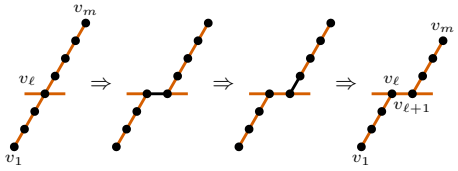
1. If $\ell < r$, we shear the subsegment $(v_\ell, \dots, v_r)$ to align it to the $x$-axis (see Figure 6a). We obtain $r - \ell + 1$ new columns. The first column consists of $(v_1, \dots, v_\ell)$. For all $i \in \{2, \dots, r - \ell\}$, the $i$-th column consists of $v_{\ell-1+i}$. The last column consists of $(v_r, \dots, v_m)$. For each of these columns, $\ell = r$ holds. Note that if $\ell = 1 \wedge r = m$, all resulting columns will consist of a single amoebot, respectively.
2. If $\ell > r$, we shear the subsegment $(v_r, \dots, v_\ell)$ twice to first align it to the $y$-axis and then to the $x$-axis (see Figure 6b). We obtain $\ell - r + 1$ new columns. The first column consists of $(v_\ell, \dots, v_m)$. For all $i \in \{2, \dots, \ell - r\}$, the $i$-th column consists of $v_{\ell+1-i}$. The last column consists of $(v_1, \dots, v_r)$. For each of these columns, $\ell = r$ holds. Note that if $\ell = m \wedge r = 1$, all resulting columns will consist of a single amoebot, respectively.
3. If $\ell = r$, we apply the following subrules.
   a. If $1 < \ell < m$, we split the column into two columns as follows (see Figure 6c). First, $v_\ell$ expands horizontally. Second, $v_\ell$ and $v_{\ell+1}$ perform a handover. Finally, $v_{\ell+1}$ contracts again. The left column consists of $(v_1, \dots, v_\ell)$. The right column consists of $(v_{\ell+1}, \dots, v_m)$.
   b. If $\ell = 1$ and $m$ is odd, we split the column into two columns analogously to Rule 3a (see Figure 6d). The left column consists of $v_1$. The right column consists of $(v_2, \dots, v_m)$.



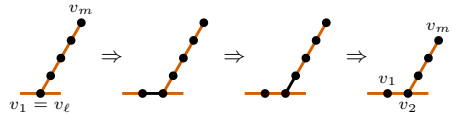

**(a)** Rule 1 ($\ell < r$).



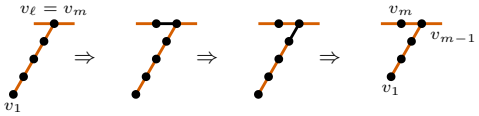

**(b)** Rule 2 ($\ell > r$).



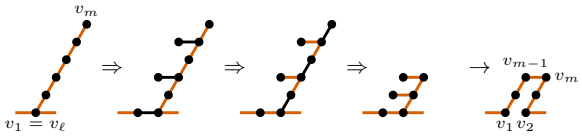

**(c)** Rule 3a ($\ell = r$ and $1 < \ell < m$).



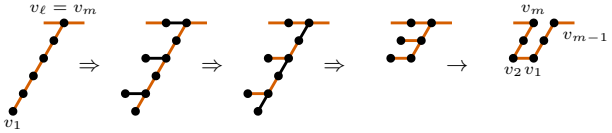

**(d)** Rule 3b ($\ell = r$, $\ell = 1$ and $m$ odd).




**(e)** Rule 3c ($\ell = r$, $\ell = m$ and $m$ odd).




**(f)** Rule 3d ($\ell = r$, $\ell = 1$ and $m$ even).




**(g)** Rule 3e ($\ell = r$, $\ell = m$ and $m$ even).

**Figure 6** Rules used in the Monotone2LineSegment algorithm.

**c.** If $\ell = m$ and $m$ is odd, we split the column into two columns analogously to Rules 3a and 3b (see Figure 6e). The left column consists of $v_m$. The right column consists of $(v_2, \dots, v_{m-1})$.

**d.** If $\ell = 1$ and $m$ is even, we split the subsegment $(v_\ell, \dots, v_m)$ into two columns with $m/2$ amoebots, respectively, as follows (see Figure 6f). For each $i \in \{1, 3, \dots, m-1\}$, $v_i$, we perform the following procedure in parallel. In the first round, each $v_i$ expands horizontally. In the second round, each $v_i$ performs a handover with $v_{i+1}$. In the third round, each $v_{i+1}$ contracts. We obtain two columns with $m/2$ amoebots. The left column consists of $(v_1, v_3, \dots, v_{m-1})$. The right column consists of $(v_2, v_4, \dots, v_m)$.
For subsequent rule applications, we connect all amoebots within the same column with bonds and remove all bonds between those columns except for the topmost horizontal bond. Note that this does not require an additional round.

**e.** If $\ell = m$ and $m$ is even, we split the subsegment $(v_\ell, \dots, v_m)$ into two columns with $m/2$ amoebots, respectively, analogously to Rules 3d (see Figure 6g). The left column consists of $(v_2, v_4, \dots, v_m)$. The right column consists of $(v_1, v_3, \dots, v_{m-1})$.
For subsequent rule applications, we connect all amoebots within the same column with bonds and remove all bonds between those columns except for the bottommost horizontal bond. Note that this does not require an additional round.

We obtain the following theorem.

▶ **Theorem 6.** *The Monotone2LineSegment algorithm reconfigures a structure of* Monotone *into a structure of* LineSegment *in* 16 *rounds.*

In the full version, we also show how the Monotone2LineSegment algorithm can be used to reconfigure a star-convex structure into a monotone structure.

▶ **Theorem 7.** *The StarConvex2Monotone algorithm reconfigures a structure of* StarConvex *into a structure of* Monotone *in* $O(1)$ *rounds.*

## 5 Universal Reconfiguration

In this section, we describe our universal reconfiguration algorithm that reconfigures an arbitrary structure into a line segment. It consists of 3 subroutines. In the first subroutine, we apply an algorithm (called Arbitrary2Bounded) to reconfigure the structure into a bounded structure. In the second subroutine, we apply an algorithm (called Bounded2Monotone) to reconfigure the bounded structure into a monotone structure. Finally, in the third subroutine, we apply the Monotone2LineSegment algorithm to reconfigure the monotone structure into a line segment. The latter subroutine was presented in Section 4. In the following, we present the other two subroutines.

First, consider the **reconfiguration into a bounded structure**. W.l.o.g., we will bound the number of rows to $\sqrt{n}$. The idea is to iteratively merge rows with less than $\sqrt{n}$ amoebots with neighboring rows. Each iteration proceeds as follows. We first pair a maximum number of adjacent rows such that each pair has at least one row with less than $\sqrt{n}$ amoebots.

Then, we merge each pair of rows as follows. The merge consists of 3 phases. In the first phase, each amoebot expands into the other row parallel to the $y$-axis if the node is unoccupied (see Figure 7). Observe that the amoebots that are not able to expand form pairs (see the blue amoebots in Figure 7).

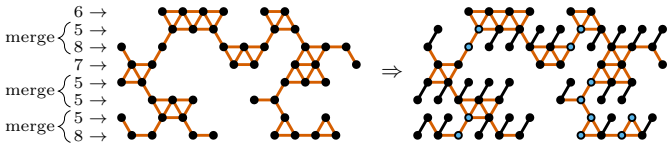
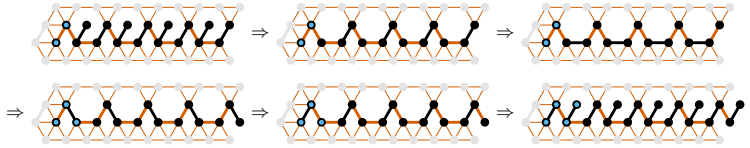
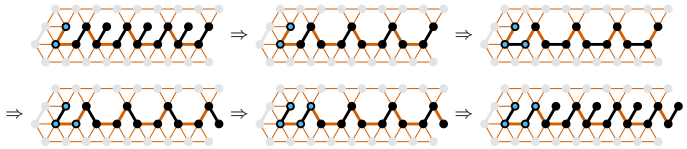



**Figure 7** First phase of the Arbitrary2Bounded algorithm. The amoebot structure has $n = 49$. Hence, each pair of rows that we want to merge has at least one row with less than $\sqrt{n} = 7$ amoebots. The blue amoebots indicate the pairs of amoebots that are unable to expand.



**(a)** Even number of amoebots.



**(b)** Odd number of amoebots.

**Figure 8** Single iteration of the second phase of the Arbitrary2Bounded algorithm. The blue amoebots indicate a pair of amoebots that were unable to expand in the first phase. The gray nodes are initially adjacent to the participating amoebots.

In the second phase, we iteratively move one amoebot of each of these pairs to the next unoccupied position in the row. W.l.o.g., consider the rightmost pair of amoebots and all expanded amoebots to the next unoccupied position to the right (see Figure 8). The idea is to construct an alternating chain from the pair of amoebots to the next unoccupied position. For that, we contract every second amoebot to its bottom node, ensuring the rightmost amoebot remains expanded. This construction guarantees that we can maintain connectivity within the amoebot structure since the participating amoebots can keep at least one bond to each of the initially adjacent nodes (see the gray nodes in Figure 8). Now, we can apply the tunneling primitive to move all amoebots to the right (see Lemma 1). Finally, we expand all contracted amoebots again. We continue with the next iteration if there is another pair of amoebots. Otherwise, we continue to the next phase.

In the third phase, we contract all expanded amoebots again (see Figure 9). This concludes the merge. We obtain the following lemma.

▶ **Lemma 8.** *The Arbitrary2Bounded algorithm reconfigures a structure of* Arbitrary *into a structure of* Bounded($\sqrt{n}$) *in* $O(\sqrt{n} \log n)$ *rounds.*

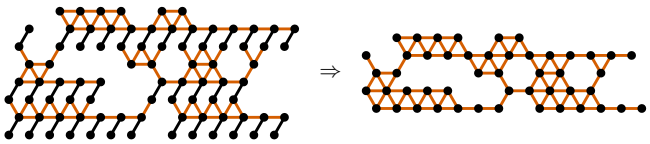


**Figure 9** Third phase of the Arbitrary2Bounded algorithm. All amoebots contract again.

**Proof.** We first show that the algorithm does not cause any collisions. In the first phase, amoebots only expand into unoccupied nodes. In the second phase, amoebots only perform isolated handovers. In the third phase, we contract whole rows. Hence, neither phase can cause a collision. Since the algorithm does not cause any collisions, the correctness follows from the fact that there can be at most $\sqrt{n}$ rows with at least $\sqrt{n}$ amoebots.

Since we can pair all rows with less than $\sqrt{n}$ amoebots (except for at most 1), the number of rows with less than $\sqrt{n}$ amoebots gets reduced by (approximately) $\frac{1}{2}$. Therefore, after $O(\log n)$ iterations, all rows have at least $\sqrt{n}$ amoebots such that we terminate.

Each iteration consists of three phases. The first and third phase only consist of a single round, respectively. A single iteration of the second phase requires 5 rounds: 1 round to construct the alternating chain, 3 rounds to apply the tunneling primitive (see Lemma 1), and 1 round to expand the amoebots again.

It remains to bound the number of necessary iterations in the second phase. Since at least one row of each pair of rows has less than $\sqrt{n}$ amoebots, there can only be at most $\sqrt{n}-1$ many pairs of amoebots that cannot expand in the first phase. Therefore, we need at most $\sqrt{n}-1$ iterations in the second phase. Overall, we require $O(\sqrt{n}\log n)$ rounds. ◀

▶ Remark 9. The second phase can be parallelized within each row. However, if all pairs of amoebots are next to each other, we can only tunnel the two outermost pairs at the same time. In the worst case, the parallelization can only speed up the second phase of the algorithm by a factor of 2. Therefore, the runtime does not improve asymptotically.

Second, consider the **reconfiguration into a monotone structure**. For that, we simulate the combing algorithm by Aloupis *et al.* [4]. We obtain the following lemma.

▶ **Lemma 10.** *The Bounded2Histogram algorithm reconfigures a structure of* Bounded($k$) *into a structure of* Monotone *in* $O(k)$ *rounds.*

Finally, the universal reconfiguration algorithm applies the Monotone2Line algorithm to reconfigure the monotone structure to a line segment (see Section 4). By combining Lemmas 8 and 10 and Theorem 6, we obtain the following theorem.

▶ **Theorem 11.** *The universal reconfiguration algorithm reconfigures a structure of* Arbitrary *into a structure of* LineSegment *in* $O(\sqrt{n}\log n)$ *rounds.*

# 6 Spirals

In this section, we present two algorithms to reconfigure a spiral into a line. In Section 6.1, we describe a simple algorithm that preserves the order of the amoebots within the spiral. Preserving the order may be interesting when the amoebots store information. In Section 6.2, we show a faster algorithm that only requires constant time.

## 6.1 Unrolling

In this section, we present our unrolling algorithm to reconfigure a spiral into a line while preserving the order of the amoebots. The algorithm iteratively removes corners by aligning the outermost segment to the next segment. For that, we simply apply the shearing primitive. We terminate once all corners were removed. We obtain the following theorem.

▶ **Theorem 12.** *The unrolling algorithm reconfigures a structure of* Spiral *into a structure of* LineSegment *while preserving the order of the amoebots in* $O(c)$ *rounds where* $c$ *denotes the number of corners.*

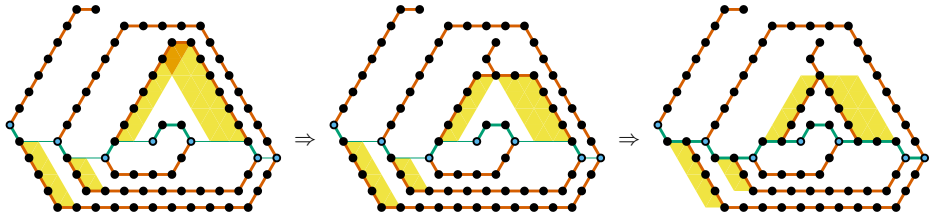


**Figure 10** Base line construction. The blue amoebots indicate the left and right corners. The green path indicates the base line. The yellow areas indicate the parallelograms. The orange areas indicate the areas where parallelograms intersect. Each double arrow indicates multiple rounds.

▶ Remark 13. We cannot accelerate the algorithm by applying the shearing primitive on multiple segments in parallel because the number of shearing primitives the initially outermost segment has to participate in is linear in the number of corners. Furthermore, parallelization can lead to collisions.

## 6.2 Constant-Time Algorithm

In this section, we present our Spiral2LineSegment algorithm. We assume that the spiral consists of at least 5 line segments. Otherwise, we resort to the unrolling algorithm which gives us a constant-time reconfiguration in this case. W.l.o.g., we assume that the spiral starts with a $y$-segment, spirals outwards in a clockwise direction, and ends with a sufficiently long $x$-segment. By sufficiently long, we mean that it is long enough to perform a parallelogram primitive if necessary (see below). We may perform up to 4 iterations of the unrolling algorithm to enforce the latter assumption.

Similar to the universal reconfiguration algorithm, we go through a monotone structure. Hence, the algorithm starts with the Spiral2Monotone algorithm which reconfigures the spiral into a $y$-monotone structure. The idea is to form a base line through the spiral which allows us to cut the "arcs" of the spiral without disconnecting the structure. In order to then obtain a monotone structure, we only need to align all segments that are not part of the base line, w.l.o.g., with the $y$-axis. In the following, we explain both parts in more detail.

We first define a *base line* as follows (see Figure 10). We start with the first three innermost segments. From each left (right) corner, we go straight to the left (right) until we hit the next line segment if such line segment exists. From there, we go to the next left (right) corner of that line segment. If such a line segment does not exist, we stop the base line at the corner. Note that the base line is $y$- and $z$-monotone by construction.

In order to construct the base line, we need to fill the paths from the left (right) corners to the next line segments to the left (right). Observe that the remaining base line is already occupied by amoebots. The idea is to apply the parallelogram primitive on each hit line segment and its adjacent $x$-segment in parallel. If the base line hits a corner, we can choose either incident line segment. However, the parallelograms may intersect with each other or may degenerate because the $x$-segments are too short. To resolve these issues, we can apply the trapezoid primitive. We choose the starting point such that it becomes an acute corner of the parallelogram.

The base line allows us to disconnect each $x$-segment (except for the innermost one, which is part of the base line) at one of its endpoints without disconnecting the amoebot structure. Note that some $x$-segments could been reduced to a single amoebot. Similar to the triangle primitive (see Lemma 4), we will call all line segments that are not part of the base line "arms". It remains to align the arms to the $y$-axis with the shearing primitive. For this, we must first ensure that we have enough space to perform all applications of the primitive.

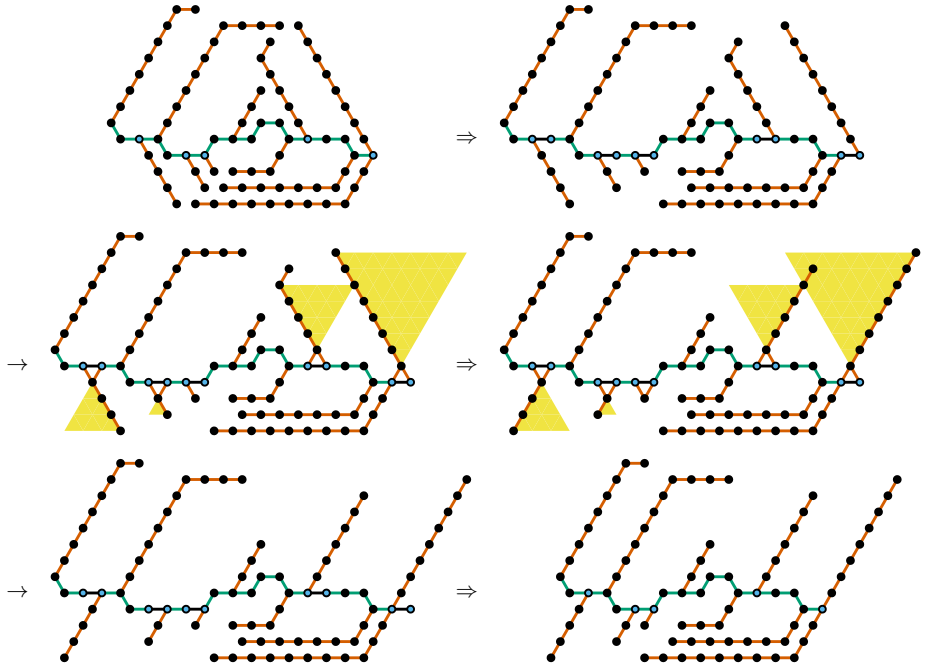


**Figure 11** Alignment of the $z$-segments. The blue amoebots indicate the amoebots on the base line connected to a $z$-segment of an arm. The green path indicates the base line. The yellow areas indicate the shearing primitive. The second double arrow indicates 2 rounds.

We start by aligning the $z$-segments in the arms in the following 3 phases (see Figure 11). In the first phase, we expand all amoebots on the base line connected to a $z$-segment of an arm horizontally. In the second phase, we can apply the shearing primitive on each $z$-segment. In the third phase, we contract all amoebots again.

Next, we align the $x$-segments in the arms in the following 3 phases (see Figure 12). In the first phase, we expand all left and right corners on the base line parallel to the $z$-axis, and all corners in the arms parallel to the $z$-axis. In the second phase, we can apply the shearing primitive on each $x$-segment. In the third phase, we contract all amoebots again.

▶ **Lemma 14.** *The Spiral2Monotone algorithm reconfigures a structure of* Spiral *into a structure of* Monotone *in* $O(1)$ *rounds.*

By combining Lemma 14 and Theorem 6, we obtain the following theorem.

▶ **Theorem 15.** *The Spiral2LineSegment algorithm reconfigures a structure of* Spiral *into a structure of* LineSegment *in* $O(1)$ *rounds.*

## 7 Open Problems

In this paper, we have shown how to reconfigure any structure into another in sublinear time. It is still open whether a polylogarithmic time universal reconfiguration algorithm is possible. One way to achieve this could be an algorithm that reconfigures a structure of Arbitrary into a structure of Bounded(polylog $n$), which would reduce the runtime of our universal reconfiguration algorithm to polylogarithmic time. Another way would be an algorithm that reconfigures a structure of Arbitrary into a structure of Rhombical for which we already know how to reconfigure it to any other structure of Rhombical in

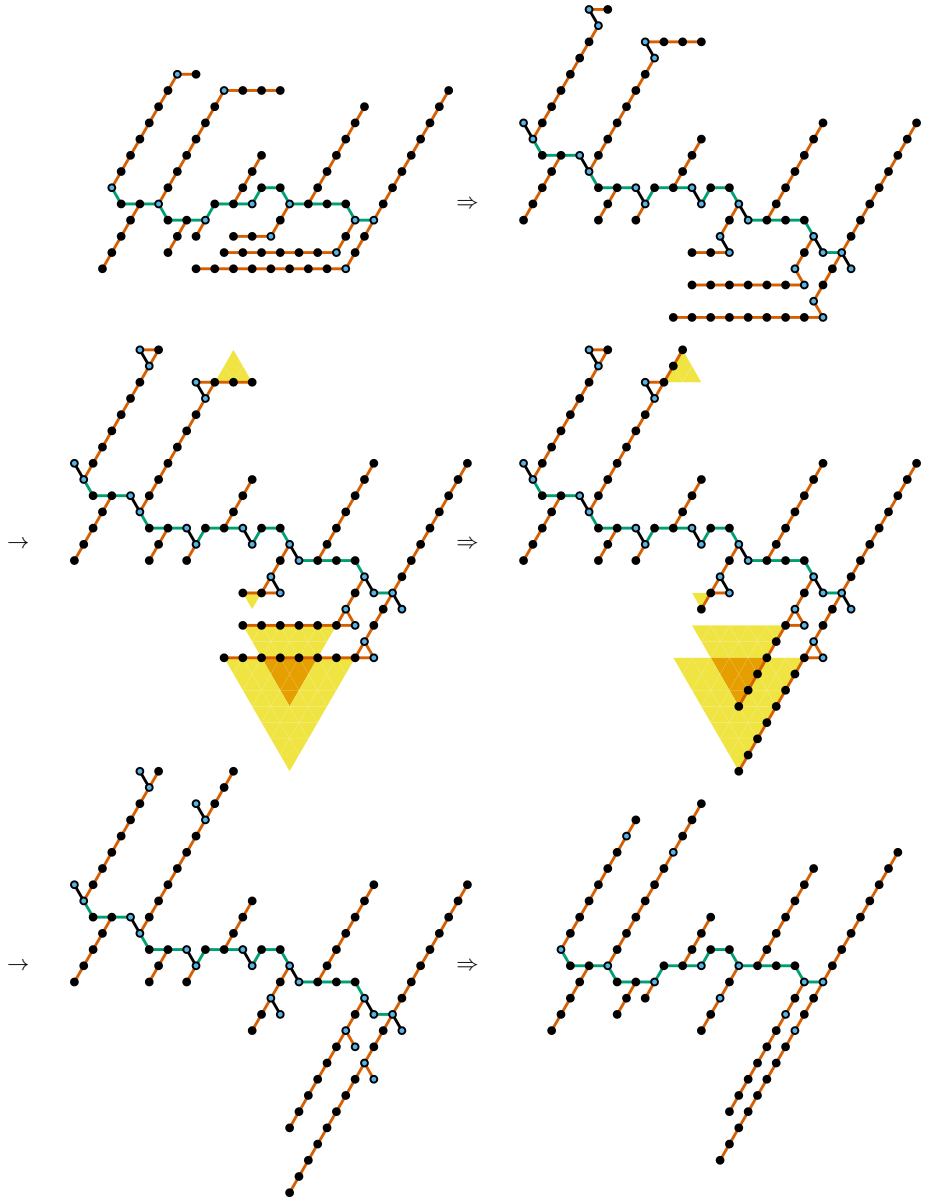


**Figure 12** Alignment of the $x$-segments. The blue amoebots indicate all left and right corners on the base line and all corners in the arms. The green path indicates the base line. The yellow areas indicate the shearing primitive. The second double arrow indicates 2 rounds.

logarithmic time [5, 15]. We have also presented an algorithm that reconfigures a spiral into a line segment in constant time. It would be interesting to investigate which other classes of structures can be reconfigured into a line segment in constant time.

Furthermore, we left the design of distributed algorithms for future work. Recall that local communication between amoebots leads to a natural lower bound of $\Omega(D)$, where $D$ is the diameter of the structure. Therefore, other ways of communication are required, e.g., the *reconfigurable circuit extension* [11] which enables fast global communication between amoebots. Previous work has shown that reconfigurable circuits enable polylogarithmic-time solutions to various stationary problems [6, 11, 16, 17]. Almalki *et al.* [1] were the first to leverage them for rapid, distributed transformations in the growth model. It is open whether they can also prove of use in the amoebot model with joint movements.

## References


1 Nada Almalki, Siddharth Gupta, Othon Michail, and Andreas Padalkin. Efficient distributed algorithms for shape reduction via reconfigurable circuits. In *SSS*, volume 16350 of *Lecture Notes in Computer Science*, pages 40–55. Springer, 2025. doi:10.1007/978-3-032-11127-2_5.

2 Nada Almalki and Othon Michail. On geometric shape construction via growth operations. *Theor. Comput. Sci.*, 984:114324, 2024. doi:10.1016/J.TCS.2023.114324.

3 Abdullah Almethen, Othon Michail, and Igor Potapov. Pushing lines helps: Efficient universal centralised transformations for programmable matter. *Theor. Comput. Sci.*, 830-831:43–59, 2020. doi:10.1016/J.TCS.2020.04.026.

4 Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D. Demaine, Robin Y. Flatland, Stefan Langerman, Joseph O'Rourke, Suneeta Ramaswami, Vera Sacristán Adinolfi, and Stefanie Wuhrer. Linear reconfiguration of cube-style modular robots. *Comput. Geom.*, 42(6-7):652–663, 2009. doi:10.1016/J.COMGEO.2008.11.003.

5 Greg Aloupis, Sébastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristán Adinolfi, and Stefanie Wuhrer. Reconfiguration of cube-style modular robots using $O(\log n)$ parallel moves. In *ISAAC*, volume 5369 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2008. doi:10.1007/978-3-540-92182-0_32.

6 Matthias Artmann, Andreas Padalkin, and Christian Scheideler. On the shape containment problem within the amoebot model with reconfigurable circuits. In *DISC*, volume 356 of *LIPIcs*, pages 7:1–7:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.DISC.2025.7.

7 Joshua J. Daymude, Kristian Hinnenthal, Andréa W. Richa, and Christian Scheideler. Computing by programmable particles. In *Distributed Computing by Mobile Entities*, volume 11340 of *Lecture Notes in Computer Science*, pages 615–681. Springer, 2019. doi:10.1007/978-3-030-11072-7_22.

8 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The canonical amoebot model: algorithms and concurrency control. *Distributed Comput.*, 36(2):159–192, 2023. doi:10.1007/S00446-023-00443-3.

9 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: amoebot - a new model for programmable matter. In *SPAA*, pages 220–222. ACM, 2014. doi:10.1145/2612669.2612712.

10 Zahra Derakhshandeh, Robert Gmyr, Thim Strothmann, Rida A. Bazzi, Andréa W. Richa, and Christian Scheideler. Leader election and shape formation with self-organizing programmable matter. In *DNA*, volume 9211 of *Lecture Notes in Computer Science*, pages 117–132. Springer, 2015. doi:10.1007/978-3-319-21999-8_8.

11 Michael Feldmann, Andreas Padalkin, Christian Scheideler, and Shlomi Dolev. Coordinating amoebots via reconfigurable circuits. *J. Comput. Biol.*, 29(4):317–343, 2022. doi:10.1089/CMB.2021.0363.

12 Ferran Hurtado, Enrique Molina, Suneeta Ramaswami, and Vera Sacristán Adinolfi. Distributed reconfiguration of 2d lattice-based modular robotic systems. *Auton. Robots*, 38(4):383–413, 2015. doi:10.1007/S10514-015-9421-8.

13 Irina Kostitsyna, Cai Wood, and Damien Woods. Turning machines: a simple algorithmic model for molecular robotics. *Natural Computing*, 23(2):407–430, 2024. doi:10.1007/S11047-022-09880-8.

14 Manish Kumar, Othon Michail, Andreas Padalkin, and Christian Scheideler. Sublinear-time reconfiguration of programmable matter with joint movements. *CoRR*, abs/2603.10720, 2026. doi:10.48550/arXiv.2603.10720.

15 Andreas Padalkin, Manish Kumar, and Christian Scheideler. Reconfiguration and locomotion with joint movements in the amoebot model. *Auton. Robots*, 49(3):22, 2025. doi:10.1007/S10514-025-10204-9.

16 Andreas Padalkin and Christian Scheideler. Polylogarithmic time algorithms for shortest path forests in programmable matter. In *PODC*, pages 65–75. ACM, 2024. doi:10.1145/3662158.3662776.

17 Andreas Padalkin, Christian Scheideler, and Daniel Warner. The structural power of reconfigurable circuits in the amoebot model. *Nat. Comput.*, 23(4):603–625, 2024. doi:10.1007/S11047-024-09981-6.

18 Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Auton. Robots*, 10(1):107–124, 2001. doi:10.1023/A:1026504804984.

19 Tommaso Toffoli and Norman Margolus. Programmable matter: Concepts and realization. *Int. J. High Speed Comput.*, 5(2):155–170, 1993. doi:10.1142/S0129053393000086.

20 Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *ITCS*, pages 353–354. ACM, 2013. doi:10.1145/2422436.2422476.