Algorithmica (2021) 83:387–412
https://doi.org/10.1007/s00453-020-00761-z

CADbots: Algorithmic Aspects of Manipulating Programmable Matter with Finite Automata

Sándor P. Fekete[1] · Robert Gmyr[2] · Sabrina Hugo[1] · Phillip Keldenich[1] · Christian Scheffer[1] · Arne Schmidt[1]

Received: 9 July 2019 / Accepted: 17 August 2020 / Published online: 4 September 2020

Abstract

We contribute results for a set of fundamental problems in the context of programmable matter by presenting algorithmic methods for evaluating and manipulating a collective of particles by a finite automaton that can neither store significant amounts of data, nor perform complex computations, and is limited to a handful of possible physical operations. We provide a toolbox for carrying out fundamental tasks on a given arrangement of particles, using the arrangement itself as a storage device, similar to a higher-dimensional Turing machine with geometric properties. Specific results include time- and space-efficient procedures for bounding, counting, copying, reflecting, rotating or scaling a complex given shape.

Keywords Programmable matter · Finite automata · Robots · Geometry · Turing machines · CAD operations

Sándor P. Fekete
fekete@ibr.cs.tu-bs.de

Robert Gmyr
robert@gmyr.net

Sabrina Hugo
hugo@ibr.cs.tu-bs.de

Phillip Keldenich
keldenich@ibr.cs.tu-bs.de

Christian Scheffer
scheffer@ibr.cs.tu-bs.de

Arne Schmidt
aschmidt@ibr.cs.tu-bs.de

1 Department of Computer Science, TU Braunschweig, Braunschweig, Germany

2 Department of Computer Science, University of Paderborn, Paderborn, Germany

1 Introduction

When dealing with classic challenges of robotics, such as exploration, evaluation and manipulation of objects, traditional robot models are based on relatively powerful capabilities, such as the ability (1) to collect and store significant amounts of data, (2) perform intricate computations, and (3) execute complex physical operations. With the ongoing progress in miniaturization of devices, new possibilities emerge for exploration, evaluation and manipulation. However, dealing with small dimensions (as present in the context of micro-robots and even programmable matter) or large dimensions (as in the construction of large-scale structures in space) introduces a vast spectrum of new difficulties and constraints. These include significant limitations to all three of the mentioned capabilities; challenges get even more pronounced in the context of complex small-scale or far-away systems, where there is a significant threshold between "internal" objects and sensors, and "external" control entities [2]. that can evaluate gathered data, extract important information, and provide guidance.

In this paper, we present algorithmic methods for evaluating and manipulating a collective of particles by agents of the most basic possible type: finite automata that can neither store significant amounts of data, nor perform complex computations, and are limited to a handful of possible physical operations. The objective is to provide a toolbox for carrying out fundamental tasks on a given arrangement of particles, such as bounding, counting, copying, reflecting, rotating or scaling a large given shape. A key idea is to use the arrangement itself as a storage device, similar to a higher-dimensional Turing machine with geometric properties.

1.1 Our Results

We consider an arrangement P of N square-shaped particles, on which a single finite-state robot can perform a limited set of operations; see Sect. 2 for a precise model description. Our goal is to develop first approaches for evaluating and modifying the arrangement by defining sequences of transformations and providing metrics for such sequences. In particular, we present the following; a full technical overview is given in Table 1.

- We give a time- and space-efficient method for determining the bounding box of P, i.e., the smallest axis-aligned box containing P.
- We show that we can simulate a Turing machine in our model.
- We provide a counter for evaluating the number N of particles forming P, as well as the number of corner pixels of P.
- We develop time- and space-efficient algorithms for higher-level operations also used in computer-aided design (CAD), such as copying, reflecting, rotating or scaling P.

Table 1 Results of this paper

Problem	Particle complexity	Time complexity
Bounding box	$\mathcal{O}(\lvert\partial P\rvert)$	$\mathcal{O}(wh\max(w,h))$
Counting		
N particles	$\mathcal{O}(\log N)^*$	$\mathcal{O}(\max(w,h)\log N + N\min(w,h))$
k Corners	$\mathcal{O}(\log k)^*$	$\mathcal{O}(\max(w,h)\log k + k\min(w,h) + wh)$
Function		
Copy	$\mathcal{O}(N)^*$	$\mathcal{O}(wh^2)$
Reflect	$\mathcal{O}(\max(w,h))^*$	$\mathcal{O}((w+h)wh)$
Rotate	$\mathcal{O}(w+h)^*$	$\mathcal{O}((w+h)wh)$
Scaling by c	$\mathcal{O}(cN)$	$\mathcal{O}((w^2+h^2)c^2N)$
Problem	**Space complexity**	**Dimension complexity**
Bounding box	$\mathcal{O}(w+h)$	$\mathcal{O}(1)$
Counting		
N particles	$\mathcal{O}(\max(w,h))$	$\mathcal{O}(1)$
k Corners	$\mathcal{O}(\max(w,h))$	$\mathcal{O}(1)$
Function		
Copy	$\mathcal{O}(wh)$	$\mathcal{O}(h)$
Reflect	$\mathcal{O}(w+h)$	$\mathcal{O}(1)$
Rotate	$\mathcal{O}(w+h+\lvert w-h\rvert\max(w,h))$	$\mathcal{O}(\lvert w-h\rvert+1)$
Scaling by c	$\mathcal{O}(c^2wh)$	$\mathcal{O}(c(w+h))$

N is the number of particles in the given shape P, w and h its width and height, and ∂P denotes the boundary of P. (*) is the particle complexity after constructing the bounding box. Particle Complexity denotes the maximum number of particles placed on the grid minus N during construction, time complexity denotes the total number of steps needed, space complexity denotes the total number of visited pixels outside the bounding box of P, and dimension complexity denotes the total number of rows and column visited outside the bounding box of P

1.2 Related Work

Practical motivation for our work arises both at very small and very large dimensions. See the overview in [2] for a discussion of work on particle computation and programmable matter, and [1, 24] for a description of possible applications for building large-scale structures in space.

There have also been a number of different, related basic models. Derakhshandeh et al. [9] introduced a fundamental concept for studying algorithmic approaches for extremely simple robots, called the *Amoebot model*. In this model, active particles move on hexagonal cells by expanding (and thus occupying two cells) and contracting. With only a few rules governing how the particles have to move, the particles can form shapes like lines, triangles or hexagons [11]. By first performing a leader election, further operations are possible [6, 14]. An algorithm for universal shapes formation was presented by Di Luna et al. [15] and by Derakhshandeh et al. [12]. The problem of coating an arbitrarily shaped object by particles was solved by

Derakhshandeh et al. [13] and analyzed in [10]. Further models involving active particles only were introduced by Woods et al. [35] (*Nubot model*) and by Hurtado et al. [20] (modular robots). Other related work includes shape formation in a number of different models, such as agents walking on DNA-based shapes [28, 32, 34], or variants of population protocols [23].

The setting of a finite-automaton robot (as an active particle) operating on a set of passive particles in a grid, called the *hybrid model*, was introduced in [19], where the objective is to arrange a given set of particles into an equilateral triangle. An extension studies the problem of recognizing certain shapes [18]. We use a simplified variant of the model underlying this line of research that exhibits three main differences: First, for ease of presentation we consider a square grid instead of a triangular grid. Second, our model is less restrictive in that the robot can create and destroy particles at will instead of only being able to transport particles from one position to another. It appears straightforward to adapt all our algorithms to the case where a robot has to bring a particle to and from a location from and to a particle depot, albeit at potentially considerable overhead for additional travel time. Finally, we allow the robot to move freely throughout the grid instead of restricting it to move along the particle structure. As shown in [24], we are able to adapt all our presented algorithms to maintain connectivity during the runtime, provided that we are allowed to use at least two robots or one robot and a special marker.

Models based on automate and movable objects have also been studied in the context of one-dimensional arrays, e.g., pebble automata [29]. Work focusing on a setting of robots on graphs includes network exploration [5], maze exploration [3], rendezvous search [27], intruder caption and graph searching [4, 17], and black hole search [22]. For a connection to other approaches to agents moving tiles, e.g., see [7, 31].

Another widely considered model considers self-assembling DNA tiles (e.g., [8, 26]) that form complex shapes based on the interaction of different glues along their edges; however, no active agents are involved, and composition is the result of chemical and physical diffusion. Although the complexity of our model is very restricted, actually realizing such a system, for example using complex DNA nanomachines, is currently still a challenging task. However, on the practical side, recent years have seen significant progress towards realizing systems with the capabilities of our model. For example, it has been shown that nanomachines have the ability to act like the head of a finite automaton on an input tape [28], to walk on a one- or two-dimensional surface [21, 25, 34], and to transport cargo [30, 32, 33].

2 Preliminaries

We consider a single *robot* that acts as a *deterministic finite automaton*. The robot moves on the (infinite) *grid* $G = (\mathbb{Z}^2, E)$ with edges between all pairs of nodes that are within unit distance using Manhattan metric. The nodes of G are called *pixels*. Each pixel is in one of two *states*: It is either *empty* or *occupied*. A *particle* denotes an occupied pixel. A *diagonal pair* is a pair of two pixels $(x_1, y_1), (x_2, y_2)$ with $|x_1 - x_2| = |y_1 - y_2| = 1$ (see Fig. 1, left and middle). A *polyomino* is a connected

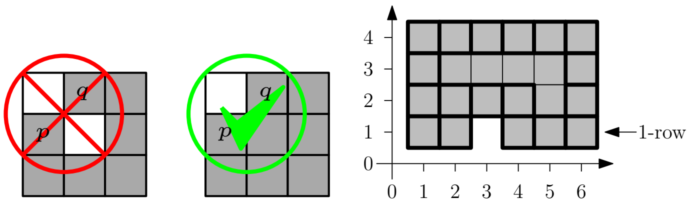


Fig. 1 Left: an illegal diagonal pair (p, q). Middle: An allowed diagonal pair (p, q). Right: a polyomino P with its boundary ∂P (particles with bold outline). P is x-monotone but not y-monotone, because the 1-row is not connected. Particles at position (2,2) and (4,2) are both building a diagonal pair with an empty pixel at position (3,1) and therefore also belong to ∂P

set of particles $P \subset \mathbb{Z}^2$ such that for all diagonal pairs $p_1, p_2 \in P$ there is another particle $p \in P$ that is adjacent to p_1 and adjacent to p_2 (see Fig. 1 middle and right). We say that P is *simple* if it has no holes, i.e., if $G \setminus P$ is connected. Otherwise, P is *non-simple*.

The *a-row* of P is the set of all pixels $(x, a) \in P$. We say that P is *y-monotone* if the a-row of P is connected in G for each $a \in \mathbb{Z}$ (see Fig. 1 right). Analogously, the *a-column* of P is the set of all pixels $(a, y) \in P$ and P is called *x-monotone* if the a-column of P is connected in G for each $a \in \mathbb{Z}$. The *boundary* ∂P of P is the set of all particles of P that are adjacent to an empty pixel or that build a diagonal pair with an empty pixel (see Fig. 1 right). The bounding box of a given polyomino P is defined as the smallest axis-aligned rectangle enclosing P.

A *configuration* consists of the states of all pixels and the robot's location and state. Initially, the robot is placed on a particle. We assume that the robot uses a local compass to distinguish the four directions the robot can move to. Because we are only using one robot, we may assume that this local compass matches the global compass. The robot can transform a configuration into another configuration using a sequence of look-compute-move steps as follows. In each step, the robot acts according to a *transition function* δ. This transition function maps a pair (p, b) containing the state of the robot and the state of the current pixel to a triple (q, c, d), where q is the new state of the robot, c is the new state of the current pixel, and $d \in \{\text{up}, \text{down}, \text{left}, \text{right}\}$ is the direction the robot moves in. In other words, in each step, the robot checks its state p and the state of the current pixel b, computes $(q, c, d) = \delta(p, b)$, changes into state q, changes the state of the current pixel to c if $c \neq b$, and moves one step into direction d.

Our goal is to develop robots transforming an unknown initial configuration P into a target configuration $T(P)$ by moving, creating, and deleting particles. We assess the efficiency of a robot by several metrics:

- *Time complexity* The total number of steps performed by the robot until termination.
- *Dimension complexity* The number of rows and columns we visit outside the bounding box of P. (Having this complexity can help having multiple robots working on different polyominoes at the same time, e.g. by separating the polyominoes by enough space so the robots do not use workspace of other robots.)

- *Space complexity* the total number of visited pixels outside the bounding box of the input polyomino *P*. (This gives a slightly more precise analysis of the used space, e.g., if we need only $O(1)$ extra rows and columns.)
- *Particle complexity* The maximum number of particles on the grid minus the number of particles in the input polyomino *P* at any given time. (The idea is that, once created, particles can be stored at some place, where robots can pick up and store particles whenever needed, and thus, already created particles can be reused. Carrying out these operations efficiently is a separate algorithmic problem that is not at the heart of our presented work and will be dealt with separately.)

3 Basic Tools

A robot can check the states of all pixels within a constant distance by performing a tour of constant length. Thus, from now on we assume that a robot can check within a single step all eight pixels that are adjacent to the current pixel *p* or build a diagonal pair with *p*.

3.1 Bounding Box Construction

Because the bounding box and the polyomino are comprised of identical tiles, we need a gap between both to differentiate the two structures. We describe how to construct a bounding box of a given, not necessarily simple polyomino *P* in the checkered pattern shown in Fig. 2. We can split the bounding box into an *outer lane* and an *inner lane* (see Fig. 2). This allows distinguishing particles of *P* and particles of the bounding box. We assume that the robot starts anywhere on *P*.

Our construction proceeds in three phases. (i) We search for an appropriate starting position. (ii) We wrap a checkered path around *P*. (iii) We finalize this path to obtain the bounding box.

For **phase (i)**, we simply search for a local minimum in the *y*-direction:

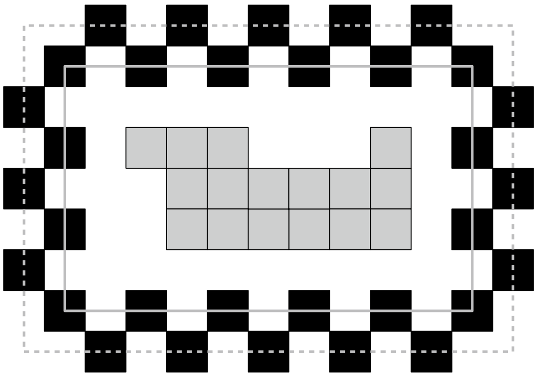


Fig. 2 A Polyomino *P* (gray) surrounded by a bounding box of particles (black) on the inner lane (solid line) and particles on the outer lane (dashed line)

1. Move to the leftmost particle, i.e., move left until the next pixel is empty.
2. Move to the right until there is a particle to the bottom. If there is no such particle move back to the leftmost particle, i.e., the leftmost local minimum, and end this search.
3. Move down until there is no more particle below the current position. Go back to step 1.

From the local minimum, we go two steps further to the left. If we land on a particle, this particle belongs to P and we restart phase (i) from this particle. Otherwise we have found a possible *starting position*.

In **phase (ii)**, we start constructing a path that will wrap around P. We start placing particles in a checkered pattern in the upwards direction. While constructing the path, three cases may occur.

1. At some point we lose contact with P, i.e., there is no particle at distance two from the inner lane. In this case, we do a right turn and continue our construction in the new direction.
2. Placing a new particle produces a conflict, i.e., the particle shares a corner or a side with a particle of P. In this case, we shift the currently constructed side of the bounding box outwards until no more conflict occurs. This shifting process may lead to further conflicts, i.e., we may not be able to further shift the current side outwards. If this happens, we deconstruct the path until we can shift it outwards again (see Fig. 3). In this process, we may be forced to deconstruct the entire bounding box we have built so far, i.e., we remove all particles of the bounding box including the particle in the starting position. In this case we know that there must be a particle of P to the left of the start position; we move to the left until we reach such a particle and restart phase (i).
3. Placing a new particle closes the path, i.e., we reach a particle of the bounding box we have created so far. We proceed with phase (iii).

Phase (iii) Let t be the particle that we reached at the end of phase (ii). We can distinguish two cases: (i) At t, the bounding box splits into three different

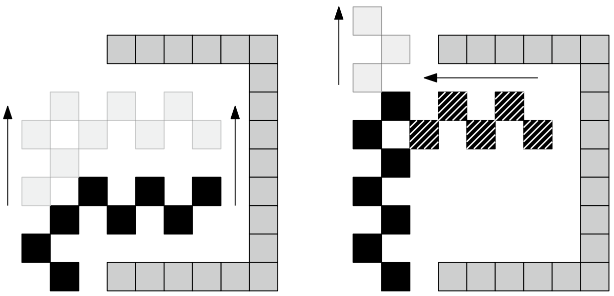


Fig. 3 Left: further construction is not possible. Therefore, we shift the line upwards (see light gray particles). Right: a further shift produces a conflict with the polyomino. Thus, we remove particles (diagonal stripes) and proceed with the shift (light gray) when there is no more conflict

directions (shown as the particle with black-and-white diagonal stripes in Fig. 4), and (ii) the bounding box splits into two different directions. In the latter case we are done, because we can only reach the bottom or left side of the bounding box. In the first case, we can move from t to the particle where we started the bounding box construction by doing a right turn and following the checkered path, and remove the bounding box parts until we reach t again. Now the bounding box splits in two directions at t. However, we may not have built a convex shape around P (see Fig. 4 left). This can be dealt with by straightening the bounding box in an analogous fashion like shifting in phase (2), e.g., by pushing the line to the right of t down.

Theorem 1 *The described strategy builds a bounding box that encloses the given polyomino P of width w and height h. Moreover, the strategy terminates after $\mathcal{O}(\max(w,h) \cdot wh)$ steps, using at most $\mathcal{O}(|\partial P|)$ auxiliary particles and $\mathcal{O}(w+h)$ of additional space in $O(1)$ extra columns and rows. The running time in the best case is $\mathcal{O}(wh)$.*

Proof **Correctness** We show that (a) the bounding box will enclose P, and (b) whenever we make a turn or shift a side of the bounding box, we find a particle with distance two from the bounding box, i.e., we will not build an infinite line.

First note that we only make a turn after encountering a particle with distance two to the inner lane. This means that we will "gift wrap" the polyomino until we reach the start. When there is a conflict (i.e., we would hit a particle of P with the current line), we shift the current line. Thus, we again find a particle with distance two to the inner lane. This also happens when we remove the current line and extend the previous one. After a short extension we make a turn and find a particle with distance two to the inner lane, meaning that we make another turn at some point. Therefore, we do not construct an infinite line, and eventually close the loop at some point.

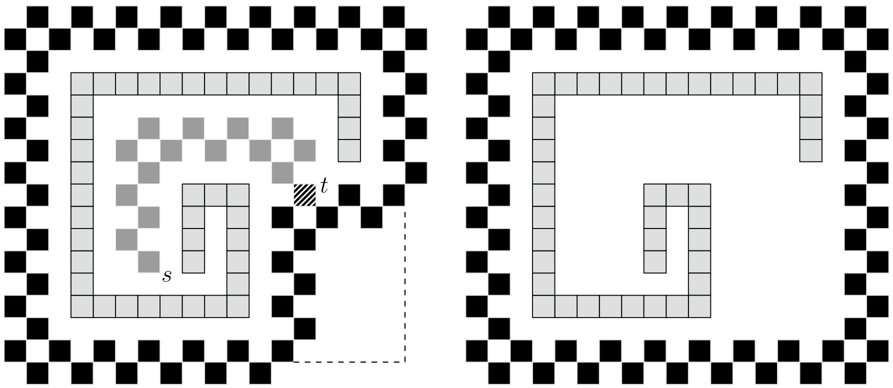


Fig. 4 Left: during construction, starting in s, we reach a particle t (diagonal stripes) at which the bounding box is split into three directions. The part between s and t (dark gray particles) can be removed, followed by straightening the bounding box along the dashed line. Right: the final bounding box

Time Every time we enter phase (1), we visit each particle of P at most $\mathcal{O}(1)$ times. When comparing to runs of phase (1), we observe that no particle of P is visited in both runs. Therefore, phase (1) costs $\mathcal{O}(N)$ time for the whole algorithm.

To establish a runtime of $\mathcal{O}(wh)$ for phase (2), we show that each pixel lying in the final bounding box is visited only a constant number of times. Consider a particle t that is visited for the first time. We know that t gets revisited again if the line through gets shifted or removed. When t is no longer on the bounding box, we can visit t again while searching for the start particle. Thus, t is visited at most four times by the robot, implying a running time of $\mathcal{O}(wh)$ unit steps. However, it may happen that we have to remove the bounding box completely and have to restart the local minimum search. In this case, there may be particles that can be visited up to $\max(w, h)$ times (see Fig. 5). Therefore, the running time is $\mathcal{O}(\max(w, h) \cdot wh)$ in the worst-case.

In phase (3), removing the path from the starting position and the particle t costs $\mathcal{O}(\partial P)$ time. Making the bounding box convex can cost at most $\mathcal{O}(wh)$ time.

In total, the strategy has a runtime of $\mathcal{O}(\max(w, h) \cdot wh)$ in the worst case, and $\mathcal{O}(wh)$ in the best case.

Auxiliary particles We now show that we need at most $\mathcal{O}(|\partial P|)$ many auxiliary particles at any time. Consider a particle t of ∂P from which we can shoot a ray to a particle of the bounding box (or the intermediate construction), such that no other particle of P is hit. For each particle t of ∂P, there are at most four particles $t_1, \dots, t_4$ of the bounding box. We can charge the cost of $t_1, \dots, t_4$ to t, which is constant. Thus, each particle in ∂P has been charged by $\mathcal{O}(1)$. However, there are still particles on the bounding box that have not been charged to a particle of ∂P, i.e., particles that lie in a curve of the bounding box.

Consider a locally convex particle t, i.e., a particle at which we can place a 2×2 square solely containing t. Because the current bounding box only does a turn if a convex particle of P has been found, each turn of the bounding box can be charged to a locally convex particle. Note that for a locally convex particle there can be at most four turns. Due to the checkered shape of the bounding box, there are at most

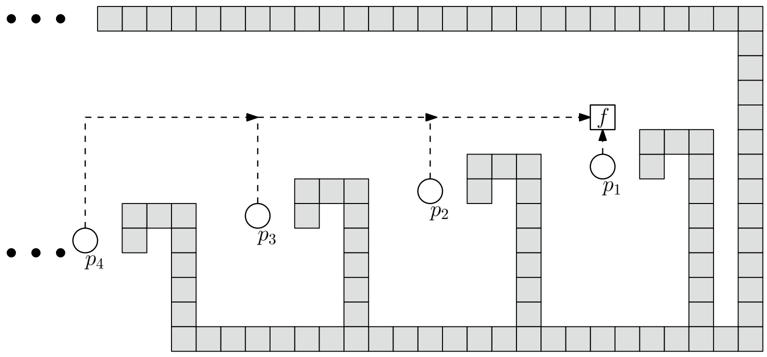


Fig. 5 A worst-case example for building a bounding box. The positions p_1 to p_4 denote the first, second, third and fourth starting position of the robot during bounding box construction. With each restart, the pixel f is visited at least once. Therefore, f can be visited $\Omega(w)$ times

four particles that are charged to a locally convex particle for each turn. Therefore, each locally convex particle gets charged by $\mathcal{O}(1)$ implying that each particle of ∂P has constant cost, i.e., we need at most $\mathcal{O}(\partial P)$ auxiliary particles.

Because we never move further away from the polyomino than the resulting bounding box, we need $\mathcal{O}(w+h)$ additional space in $\mathcal{O}(1)$ extra rows and columns. □

Even if the starting configuration P contains a forbidden configuration, i.e., there is a diagonal pair of particles having no common adjacent particle, we can construct the bounding box in the same way. However, the test if a particle belongs to P or to the bounding box is slightly different: Let p_1, p_2 be a diagonal pair with no common adjacent tile. Then, p_1 and p_2 have at least one adjacent particle p_3 and p_4, resp. Therefore, the robot can perform a local lookup if one of p_3 or p_4 exists and we get the following observation.

Observation 2 *We can build a bounding box that encloses a given connected starting configuration P of width w and height h, i.e., a polyomino that allows forbidden configurations. Moreover, the strategy terminates after* $\mathcal{O}(\max(w,h)\cdot wh)$ *steps, using at most* $\mathcal{O}(|\partial P|)$ *auxiliary particles and* $\mathcal{O}(w+h)$ *of additional space in* $O(1)$ *extra columns and rows. The running time in the best case is* $\mathcal{O}(wh)$.

3.2 Binary Counter

For counting problems, a particle-based binary counter is indispensable, because the robot is not able to store non-constant numbers. The binary counter for storing an n-bit number consists of a base-line of n particles. Above each particle of the base-line there is either a particle denoting a 1, or an empty pixel denoting 0 (see Fig. 6). Given an n-bit counter we can *increase* and *decrease* the counter by 1 (or by any constant c), and *extend* the counter by one bit. The latter operation will only be used in an increase operation.

In order to perform an increase operation, we firstly move to the least significant bit, and secondly start flipping 1s to 0s we flip a 0 to a 1. If we have run through the counter, we extend the counter by one bit.

For the decrease operation we again start at the least significant bit, and start flipping 0s to 1s until we flip a 1 to a 0. Note that this operation only works correctly if the counter contains at least one bit set to 1.

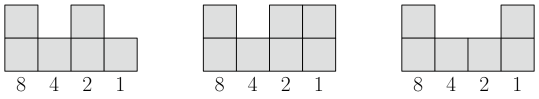


Fig. 6 Left: a 4-bit counter with decimal value 10 (1010 in binary). Middle: the binary counter increased by one. Right: The binary counter decreased by one

4 Counting Problems

The constant memory of our robot poses several challenges when we want to count certain elements of our polyomino, such as particles or corners. Because these counts can be arbitrarily large, we cannot store them in the state space of our robot. Instead, we have to make use of a binary counter. This requires us to move back and forth between the polyomino and the counter. Therefore, we must be able to find and identify the counter coming from the polyomino and to find the way back to the position where we stopped counting.

This motivates the following strategy for counting problems. We start by constructing a slightly extended bounding box and moving the robot to its bottom-left corner. From there, we perform the actual counting by shifting the polyomino two units downwards or to the left, one particle at a time. After moving each particle, we return to the counter, increasing it if necessary. We describe further details in the next two sections, where we present algorithms for counting the number of particles or corners in a polyomino.

4.1 Counting Particles

The total number of particles in our polyomino can be counted using the strategy outlined above, increasing the counter by one after each moved particle.

Theorem 3 *Let P be a polyomino of width w and height h with N particles for which the bounding box has already been created. Counting the number of particles in P can be done in $\mathcal{O}(\max(w,h)\log N + N\min(w,h))$ steps using $\mathcal{O}(\max(w,h))$ of additional space in $\mathcal{O}(1)$ extra rows and columns and $\mathcal{O}(\log N)$ auxiliary particles.*

Proof In a first step, we determine whether the polyomino's width is greater than its height or vice versa. We can do this by starting from the lower left corner of the bounding box and then alternatingly moving up and right one step until we meet the bounding box again. We can recognize the bounding box by a local lookup based on its zig-zag shape that contains particles only connected by a corner, which cannot occur in a polyomino. The height is at least as high as the width if we end up on the right side of the bounding box. In the following, we describe our counting procedure for the case that the polyomino is higher than it is wide; the other case is analogous.

We start by extending the bounding box by shifting its bottom line down by two units. Afterwards we create a vertical binary counter to the left of the bounding box. We begin counting particles in the bottom row of the polyomino. We keep our counter in a column to the left of the bounding box such that the least significant bit at the bottom of the counter is in the row in which we are currently counting. We move to the right into the current row until we find the first particle. If this particle is part of the bounding box, the current row is done. In this case, we move back to the counter, shift it upwards and continue with the next row until all rows are done.

Otherwise, we simply move the current particle down two units, return to the counter and increment it. For an example of this procedure, refer to Fig. 7.

For each particle in the polyomino, we use $\mathcal{O}(\min(w, h))$ steps to move to the particle, shift it and return to the counter; incrementing the counter itself only takes $\mathcal{O}(1)$ amortized time per particle. For each empty pixel we have cost $\mathcal{O}(1)$. In addition, we have to shift the counter $\max(w, h)$ times. Thus, we need $\mathcal{O}(\max(w, h) \log N + \min(w, h)N + wh) = \mathcal{O}(\max(w, h) \log N + \min(w, h)N)$ unit steps. We only use $\mathcal{O}(\log N)$ auxiliary particles in the counter, in addition to the bounding box.

In order to achieve $\mathcal{O}(1)$ extra rows and columns, we modify the procedure as follows. Whenever we move our counter, we check whether it extends into the space above the bounding box. If it does, we reflect the counter vertically, such that the least significant bit is at the top in the row, in which we are currently counting. This requires $\mathcal{O}(\log^2 N)$ extra steps and avoids using $\mathcal{O}(\log N)$ additional rows above the polyomino. □

4.2 Counting Corners

The counting approach described in the previous section is not restricted to counting particles. Various other features of the polyomino can be counted using the same (asymptotic) number of steps, extra particles, space and rows or columns. Essentially, the only precondition is that we must be able to identify the feature based on a local lookup. For instance, we make the following observation.

Observation 4 *Let P be a polyomino of width w and height h with k convex (reflex) corners for which the bounding box has already been created. Counting the number of convex (reflex) corners in P can be done in* $\mathcal{O}(\max(w, h) \log k + k \min(w, h) + wh)$ *steps, using* $\mathcal{O}(\max(w, h))$ *of additional space in* $\mathcal{O}(1)$ *extra rows and columns, and* $\mathcal{O}(\log k)$ *auxiliary particles.*

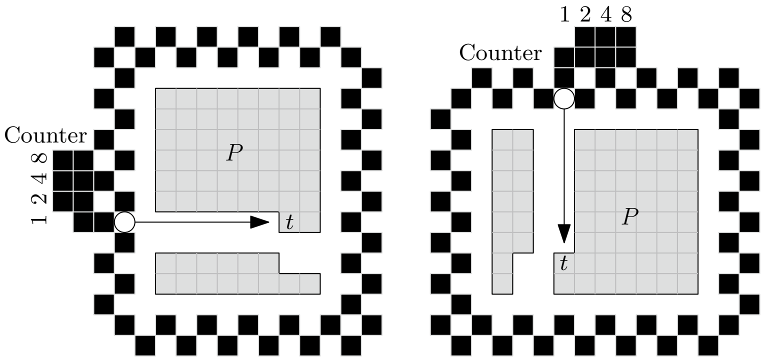


Fig. 7 An 8×8 square P in a bounding box, being counted row-by-row (left) or column-by-column (right). The robot has already counted 14 particles of P and stored this information in the binary counter. The arrow shows how the robot (○) moves to count the next particle t

Proof We use the same strategy as for counting particles in Theorem 3 and retain the same asymptotic bounds on running time, auxiliary particles and additional space and rows or columns. It remains to describe how we recognize convex and reflex corners.

Whenever we reach a particle t that we have not yet considered so far, we examine its neighbors $x_1, \dots, x_6$, as shown in Fig. 8. The particle t has one convex corner for each pair (x_1, x_2), (x_2, x_3), (x_3, x_5), (x_5, x_1) that does not contain a particle, and no convex corner is contained in more than one particle of the polyomino. As there are at most four convex corners per particle, we can simply store this number in the state space of our robot, return to the counter, and increment it accordingly.

A reflex corner is shared by exactly three particles of the polyomino, so we have to ensure that each corner is counted exactly once. We achieve this by only counting a reflex corner if it is on top of our current particle t and was not already counted with the previous particle x_1. In other words, we count the upper right corner of t as a reflex corner if exactly two of x_3, x_4, x_5 are occupied; we count the upper left corner of t as reflex corner if x_6 and x_5 are present and x_1 is not. In this way, we count all reflex corners exactly once. $\square$

5 Transformations with Turing Machines

In this section we develop a robot that transforms a polyomino P_1 into a required target polyomino P_2. In particular, we encode P_1 and P_2 by strings $S(P_1)$ and $S(P_2)$ whose characters are from $\{0, 1, \sqcup\}$ (see Fig. 9 left and Definition 1). If there is a Turing machine transforming $S(P_1)$ into $S(P_2)$, we can give a strategy that transforms P_1 into P_2 (see Theorem 5).

Note that while it may be intuitively plausible that such a relationship exists, a Turing machine itself (which works in a one-dimensional environment) does not encounter some of the issues of two-dimensional geometry, such as the more complex topology involved in determining location and boundaries. Our Theorem 5 shows that these geometric issues can be handled, paving the way for the power of

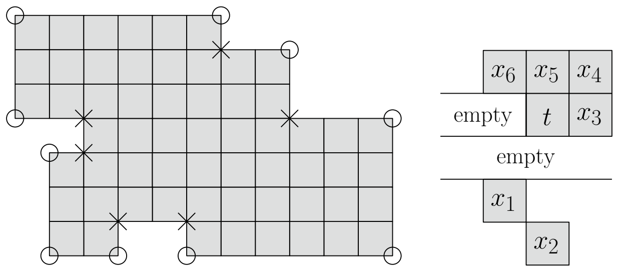


Fig. 8 Left: a polyomino and its convex corners (∘) and reflex corners (×). Right: after reaching a new particle t, we have to know which of the pixels x_1 to x_6 are occupied to decide how many convex (reflex) corners t has

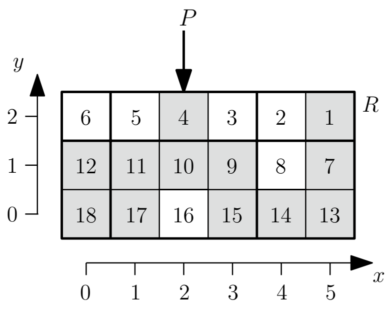


Fig. 9 An example showing how to encode a polyomino of height $h = 3$ ($S_1 = 11$ in binary) and width $w = 6$ ($S_2 = 110$ in binary) by $S(P) = S_1 \sqcup S_2 \sqcup S_3 = 11 \sqcup 110 \sqcup 100100101111111011$, where S_3 is the string of 18 bits that represent particles (black, 1 in binary) and empty pixel (white, 0 in binary), proceeding from high bits to low bits

Turing Machines. For a visualization of the basic approach, see our video [1], in particular, the part starting at 5:04.

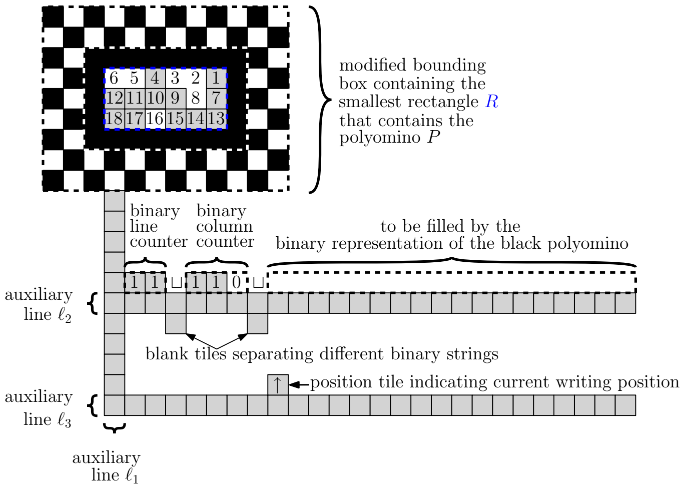
Also note that the Turing Machine emulation described in this section course also could be used, e.g., to copy polyominoes, instead of the algorithm from the next section. In practice, many applications would require a very large transition table which may not fit into the robots limited memory despite being of constant size; in these applications it is reasonable to assume that it would be more efficient w.r.t. both memory and time to use counters—and thus $\mathcal{O}(\log n)$ space—or, if the input is too large, a specialized algorithm such as the algorithms from the following Sect. 6.

We start with the definition of the encodings $S(P_1)$ and $S(P_2)$.

Definition 1 Let $R := R(P)$ be the polyomino forming the smallest rectangle containing a given polyomino P (see Fig. 9). We represent P by the concatenation $S(P) := S_1 \sqcup S_2 \sqcup S_3$ of three bit strings S_1, S_2, and S_3 separated by blanks $\sqcup$. In particular, S_1 and S_2 are the height and width of R. Furthermore, we label each pixel of R by its rank in the lexicographic decreasing order w.r.t. y- and x-coordinates with higher priority to y-coordinates (see Fig. 9). Finally, S_3 is an $|R|$-bit string $b_1, \dots, b_{|R|}$ with $b_i = 1$ if and only if the ith pixel in R is a particle of P.

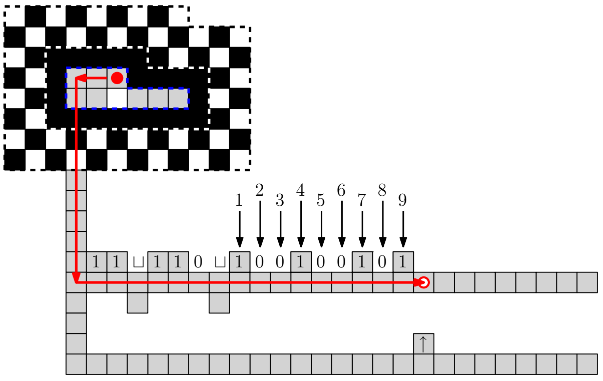
Theorem 5 *Let P_1 and P_2 be two polyominoes with $|P_1| = |P_2| = N$. There is a strategy transforming P_1 into P_2 if there is a Turing machine transforming $S(P_1)$ into $S(P_2)$. The robot needs $\mathcal{O}(\partial P_1 + \partial P_2 + S_{TM})$ auxiliary particles, $\mathcal{O}(N^4 + T_{TM})$ steps, and $\Theta(N^2 + S_{TM})$ of additional space, where T_{TM} and S_{TM} are the number of steps and additional space needed by the Turing machine.*

Proof Our strategy works in five phases: Phase (1) constructs a slightly modified bounding box of P_1 (see Fig. 10a). Phase (2) constructs a shape representing $S(P_1)$. In particular, the robot writes $S(P_1)$ onto an auxiliary line ℓ_2 in a bit-by-bit fashion (see Fig. 10b). In order to remember the position, to which the previous bit was

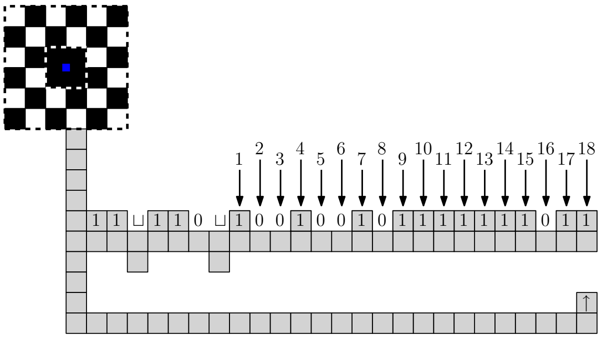


modified bounding box containing the smallest rectangle R that contains the polyomino P
6 5 4 3 2 1
12 11 10 9 8 7
18 17 16 15 14 13
binary line counter
binary column counter
to be filled by the binary representation of the black polyomino
1 1 ⊔ 1 1 0 ⊔
auxiliary line ℓ_2
blank tiles separating different binary strings
position tile indicating current writing position
auxiliary line ℓ_3
auxiliary line ℓ_1

(a) State after counting numbers of lines and columns.



(b) Intermediate state of Phase (2) while writing the string representation $S(P_1)$ of the input polyomino P_1 by placing R line by line onto ℓ_1.



(c) Final state of Phase (2) after writing the binary representation of the input polyomino.

Fig. 10 Phase (2) of transforming a (black) polyomino by simulating a Turing Machine

written, we use an additional particle called *position particle* which is not part of P_1 and placed onto another auxiliary line ℓ_3. Phase (3) simulates the Turing machine that transforms $S(P_1)$ into $S(P_2)$. Phase (4) is the reversed version of Phase (2), and Phase (5) is the reversed version of Phase (1), so we only need to discuss how to realize Phases (1), (2), and (3).

Phase (1) We apply a slightly modified version of the approach of Theorem 1. In particular, let R be the smallest rectangle enclosing the polyomino. First, we fill all pixels that lie not inside R and adjacent to the boundary of R by auxiliary particles whose union we call *shell*. Second, we construct the bounding box as described above around the previously constructed shell. This results in a third additional layer of the bounding box (see Fig. 10a). Note that the robot recognizes that the particles of the third layer do not belong to the input polyomino P, because these particles of the third layer lie adjacent to the original bounding box.

Phase (2) Initially, we construct a vertical auxiliary line ℓ_1 of constant length, seeding two horizontal auxiliary lines ℓ_2 and ℓ_3 that are simultaneously constructed by the robot during Phase (2). The robot constructs the representation of $S(P_1)$ iteratively bit-by-bit on ℓ_2 and stores on ℓ_3 a "position particle" indicating the next pixel on ℓ_2 to which the robot has to write the next bit of the representation of $S(P_1)$.

Next we apply the approach of Theorem 3 twice in order to successively construct on ℓ_2 the binary representations of the numbers of lines and the number of columns of which R is made up (see Fig. 10).

Finally, the robot places consecutively and in decreasing order w.r.t. y-coordinates all lines of R onto ℓ_2. In particular, the pixel of the current line ℓ of R are processed from right to left as follows. For each pixel $t \in \ell$, the robot alternates between ℓ and the auxiliary lines ℓ_2 and ℓ_3 in order to check whether t belongs to P_1 or not.

In order to reach the first pixel of the topmost line, the robot moves in a vertical direction on the lane induced by the vertical line ℓ_1 to the topmost line of R and then to the rightmost pixel t of the topmost line (see Fig. 10b). In order to ensure that in the next iteration of Phase (2), the next pixel of R is reached, the robot deletes a particle from t if there is a particle on pixel t from R and shrinks the modified bounding box such that it is the modified bounding box of $R \setminus \{t\}$ (see Fig. 10b). This involves only a constant-sized neighborhood of t and thus can be realized by a robot with constant memory. Next, the robot moves to the first free position of auxiliary line ℓ_2 by searching for the position particle on auxiliary line ℓ_3. As the vertical distances of ℓ_2 and ℓ_3 to the lowest line of the bounding box are of constant values, the robot can change vertically between ℓ_2 and ℓ_3 by simply remembering the robot's vertical distance to the lowest line of the bounding box. The robot concludes the current iteration of Phase (2) by moving the position particle one step to the right.

The approach of Phase (2) is repeated until all particles of P_1 are removed from R. In order to recognize that all pixels from R are removed, the robot checks whether $R = \emptyset$ in each iteration of Phase (2). This check can simply be done by checking whether the entire extended bounding box lies inside a 6×6-rectangle, as illustrated in Fig. 10c. Hence, the robot simply has to scan an environment of constant size.

Phase (3) We simply apply the algorithm of the Turing machine by moving the robot correspondingly to the movement of the head of the Turing machine.

Finally, we analyze the entire approach of simulating a Turing machine as described above. From the discussion of Phase (2) we conclude that the construction of the auxiliary lines ℓ_1, ℓ_2, and ℓ_3 are not necessary, because the vertical length of the entire construction below the bounding is a constant, i.e., in each iteration when the robot approaches the current writing position, the robot moves to the right until it arrives at the current writing position. Thus, the horizontal length of ℓ_2, ℓ_3 do not have an influence on the memory needed by the robot.

Furthermore, the current writing position on ℓ_2 is indicated by a single position particle on ℓ_3. As no auxiliary particles are needed for ℓ_1, ℓ_2, ℓ_3, the only auxiliary particles needed are the current position particle and the particles needed for the modified bounding box which lie in $\mathcal{O}(\partial P_1 + \partial P_2)$. Hence, the sizes of the bounding boxes needed in Phase (2) and Phase (4) and the additional space $\mathcal{O}(S_{TM})$ needed by the Turing machine dominate the number of used auxiliary particles, which is in $\mathcal{O}(\partial P_1 + \partial P_2 + S_{TM})$.

Furthermore, Phase (2) and Phase (4) need $\mathcal{O}(N^4)$ steps, and Phase (1) needs $\mathcal{O}(N^3)$ steps. In particular, counting the number of rows and counting the numbers of columns of the polyominoes can be done in $\mathcal{O}(N \log N + N^2)$ by using the approach of Theorem 3 because the polyominoes P_1 and P_2 may be single lines of width or height N. Furthermore, scanning a pixel, walking a distance of $\mathcal{O}(N^2)$ to the current writing position on ℓ_2, writing the current pixel, and walking to the next pixel inside the modified bounding box takes $\mathcal{O}(N^2)$ time. Hence, writing $wh \in O(N^2)$ pixels takes $\mathcal{O}(N^4)$ time. Moreover, Phase (3) needs $\mathcal{O}(T_{TM})$ steps, where T_{TM} is the number of steps needed by the Turing machine. Thus, the total time needed in the entire approach is in $\mathcal{O}(N^4 + T_{TM})$. Finally, the additional space is in $\Theta(N^2 + S_{TM})$. This concludes the proof of Theorem 5. □

6 Custom-Made Functions

As already seen, we can transform a given polyomino by any computable function by simulating a Turing machine. However, this requires a considerable amount of space. In this section, we present realizations of common operations in a more space-efficient manner, reflecting the idea that a usable toolbox for such transformations is analogous to the functions in a software package for computer-aided design (CAD).

6.1 Copying a Polyomino

Copying a polyomino P has many applications. For example, we can reproduce a given shape, or apply algorithms that may destroy P after copying P into free space. In the following, we describe the *copy* function that copies P below the bounding box in a column-wise fashion, as seen in Fig. 12 (a row-wise copy can be described analogously).

To copy a polyomino P we proceed in two phases: (1) A preprocessing step that includes building the bounding box and extending the left side of the bounding box by three units. (2) Copying pixels column-wise below the bounding box. Note that P is contained in the first $w + 1$ columns within the bounding box from the right. Therefore, we only need to start the copy procedure from the fifth column from the left.

After phase (1), we can split the bounding box into components that will be maintained by the robot: The first $k + 1$ columns $c_0, \dots, c_k$ of the bounding box from the left contain the first k columns of P already copied, then two columns c_{k+1}, c_{k+2} are used to process the next column of P, and after an empty column, the next $w - k$ columns $c_{k+4}, \dots c_{w+4}$ contain the rest of P that still needs to be copied (see Fig. 11).

We now describe how to copy the $(k + 1)$-st column of P. A problem we have to deal with is that the connected components of the intersection of one column with P may be several units apart (e.g., in Fig. 11 column c_3 has three components with one and two units space between them), and/or that the distance from the first/last particle of P to the bounding box can be arbitrarily large (e.g., in Fig. 11, column c_7 has several empty pixel until the first particle of P). Thus, we need an auxiliary structure for determining whether a pixel is empty.

We solve this by using *bridges*, i.e., auxiliary particles denoting empty pixels. To distinguish between particles of P and bridges, we use column c_{k+1} to denote particles of P, and column c_{k+2} to denote bridge particles (see Fig. 12).

To copy an empty pixel, we place a particle two unit steps to the left, i.e., in column c_{k+2}. Then we move straight down and search for the first position, at

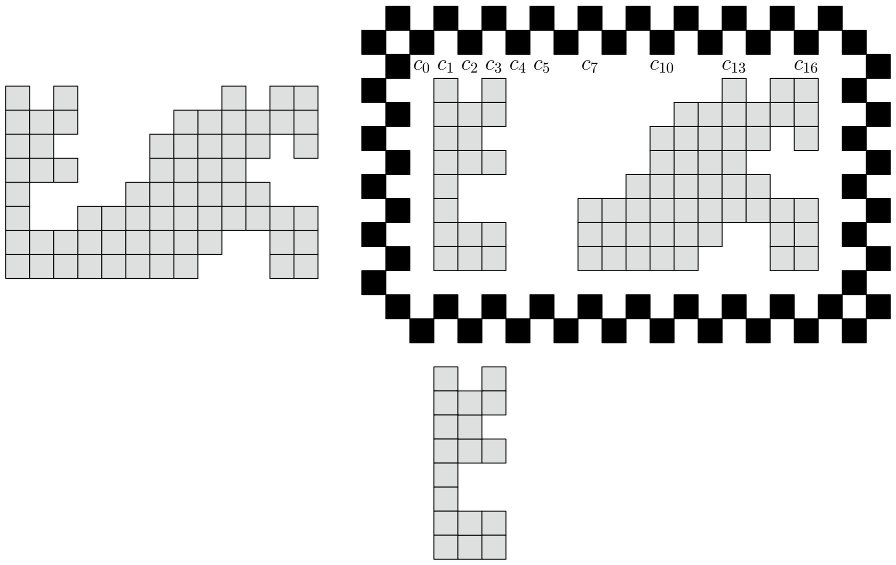


Fig. 11 Left: a polyomino P. Right: intermediate situation during the copy process. Columns c_1 to c_3 contain the first three columns of P that are already copied below the bounding box. c_4 and c_5 will be used to copy the fourth column of P. Columns c_7 to c_{16} contain particles of P that still need to be copied

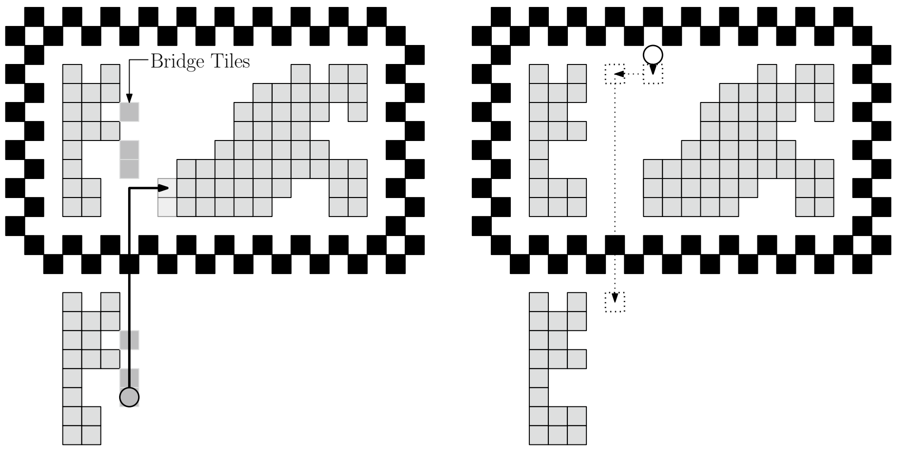


Fig. 12 Left: intermediate step while copying the third column of a polyomino (gray particles) with bridges (dark gray particles). The robot (○) moves to next pixel along the black line. Right: when the column is copied the bridges get removed and the robot proceeds with the next column. In this case the robot finds an empty pixel, places a bridge particle two steps to the left and a bridge particle below the bounding box

which there is no copied particle of P to the left, nor a bridge particle. When this position is found, we place a particle, denoting an empty position.

To copy a particle t of P, we pick up and move this particle three unit steps to the left, i.e., to column c_{k+1}. Afterwards, we move straight down following the particles and bridges until we reach a free position, i.e., there is no bridge particle to the right nor a copied polyomino particle, and place a particle, denoting a copy of t.

After placing the copy of a pixel, we move straight up until we find the first particle in c_{k+1} or c_{k+2}. From there we move two or three steps to the right (depending on whether we placed a bridge particle or not) and start to copy the next pixel. In case we reached the bottom of a column, we remove any particle representing a bridge particle and proceed with the next column (see Fig. 12 right).

Theorem 6 *Copying a polyomino P column-wise can be done within $\mathcal{O}(wh^2)$ unit steps using $\mathcal{O}(N)$ of auxiliary particles and $\mathcal{O}(wh)$ additional space in $O(h)$ extra rows and columns.*

Proof Consider the strategy described above. Because the robot copies occupied pixels and empty pixels by using bridges, the robot is always able to find the next position to place the next particle (either a copied particle of P or a bridge particle). Thus, at the end of the algorithm, we obtain a valid copy of P.

As there are $\mathcal{O}(wh)$ many pixels that we have to copy with cost of $\mathcal{O}(h)$ per pixel, the strategy needs $\mathcal{O}(wh^2)$ unit steps to terminate.

Now, consider the number of auxiliary particles. We need N particles for the copy and $\mathcal{O}(h)$ particles for bridges, which are reused for each column. Thus, we need $\mathcal{O}(N)$ auxiliary particles in total.

Because we place the copied version of P beneath P, we need $\mathcal{O}(wh)$ additional space in $O(h)$ extra rows and columns. □

6.2 Reflecting a Polyomino

In this section, we show how to perform a horizontal reflection on a polyomino P (see Fig. 13; a vertical reflection is done analogously). Assume that we already built the bounding box. Then we shift the bottom side of the bounding box one unit down, such that we have two units of space between the bounding box and P. We start with the bottom-most row r and built r in reversed order beneath the bounding box using bridges, as seen in the previous section. To do so, we use the row below r to mark empty pixels with bridge particles (see Fig. 14).

To copy the state of a pixel p, we pick up the particle (from P or from the bridge), move to the right until we reach the bounding box, move two or three steps below the bounding box (depending on whether we place a bridge particle next or not) and move to the first position where no copied particle of P nor a bridge particle exists.

To find the next pixel to copy, we move back to the right until we reach the end (i.e., the next column has no particle from P nor a bridge particle), move two steps upwards within bounding box, and search for the leftmost pixel we have not

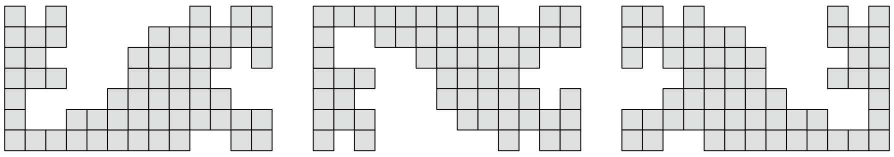


Fig. 13 Left: a polyomino P. Middle: vertical reflection of P. Right: horizontal reflection of P

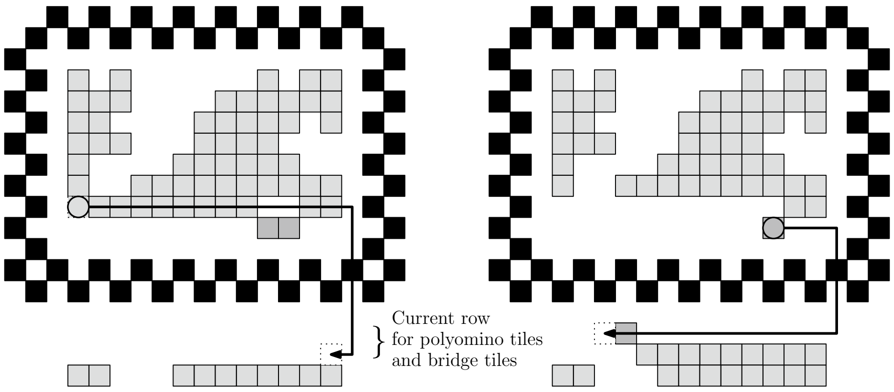


Fig. 14 Left: beginning of reflecting the second row. Because the current pixel is occupied, the robot moves the particle to the right below the bounding box (particle with dotted border). Note that the extra space between P and the bounding box is used by bridge particles (dark gray) to denote empty pixels above. Right: intermediate step of reflecting the second row. The next position is empty, thus, the robot will move the corresponding bridge particle next

copied yet. Again, this pixel is identified, when a further step has no particle from P nor a bridge particle.

After the reflection of r, we shift the bottom side of the bounding box one unit up and remove bridge particles. The rows that were reflected so far are not moved upwards, so we have again two units space below the bounding box. Also note that we again have two units of space between the bounding box and the rest of P. We can therefore repeat the process until no particle is left.

Theorem 7 *Reflecting a polyomino P horizontally can be done in $\mathcal{O}(w^2h)$ unit steps, using $\mathcal{O}(w)$ of additional space and $\mathcal{O}(w)$ auxiliary particles.*

Proof For each pixel within the bounding box we have to copy this pixel to the desired position. This costs $\mathcal{O}(w)$ steps, i.e., we have to move to the right side of the boundary, move a constant number of steps down and move $\mathcal{O}(w)$ to the desired position. These are $\mathcal{O}(w)$ steps per pixel, thus, $\mathcal{O}(w^2h)$ unit steps in total.

It can be seen in the described strategy that we only need a constant amount of additional space in one dimension because we are overwriting the space that was occupied by P. This implies a space complexity of $\mathcal{O}(w)$. Following the same argumentation of Theorem 6, we can see that we need $\mathcal{O}(w)$ auxiliary particles. □

Corollary 1 *Reflecting a polyomino P vertically and horizontally can be done in $\mathcal{O}(wh(w+h))$ unit steps, using $\mathcal{O}(w+h)$ additional space in $O(1)$ extra rows and columns and $\mathcal{O}(w+h)$ auxiliary particles.*

6.3 Rotating a Polyomino

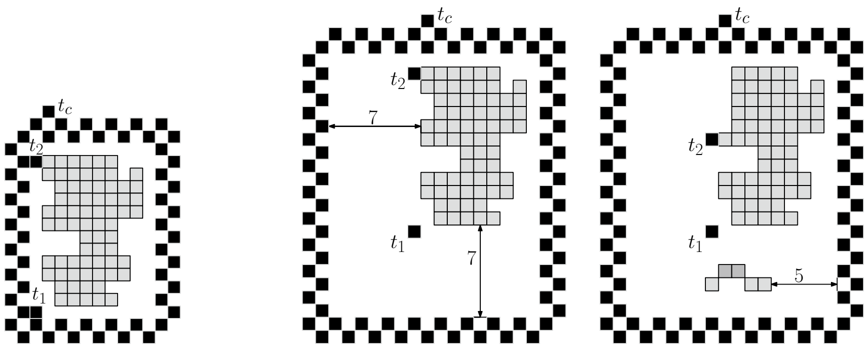
Rotating a polyomino by $\pm\frac{\pi}{2}$ presents some additional difficulties, because the dimension of the resulting polyomino has dimensions $h \times w$ instead of $w \times h$. Thus, we may need non-constant additional space in one dimension, e.g., if one dimension is large compared to the other dimension. A simple approach is to copy the rows of P bottom-up to the right of P. This allows us to rotate P with $\mathcal{O}(wh)$ additional space. For now, we assume that $h \geq w$.

We propose a strategy that is more compact. The strategy consists of two phases: First, reflect P over the $x = y$ line. Then do a second reflection over the y-axis, i.e., a horizontal reflection. Because

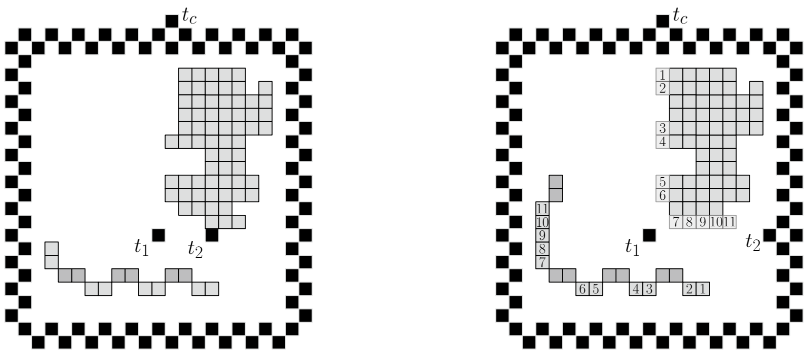
$$\underbrace{\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}}_{\substack{\text{Reflection over} \\ y\text{-axis}}} \underbrace{\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}}_{\substack{\text{Reflection over} \\ x=y\text{ line}}} P = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} P,$$

we effectively perform a clockwise rotation by $\frac{\pi}{2}$ (to perform a counter-clockwise rotation we do a vertical reflection instead of a horizontal reflection).

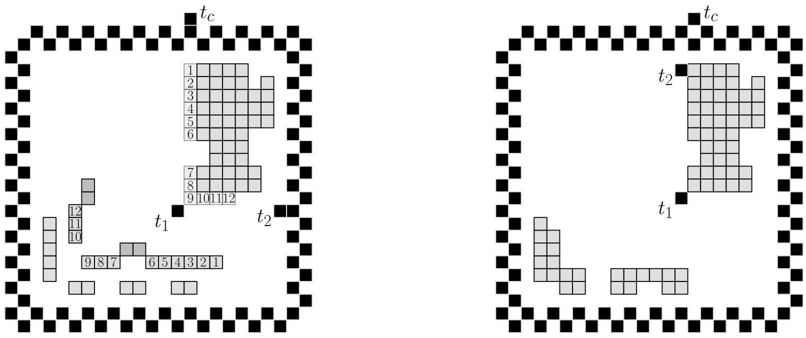
After constructing the bounding box, we place a particle t_1 in the bottom-left corner of the bounding box, which marks the bottom-left corner of P (see Fig. 15a left). We further add two more particles t_c outside the bounding box marking the position



(a) Left: Intermediate construction before expanding the bounding box. Markers t_1, t_2 and t_c are already placed. Middle: Construction after expanding the bounding box. Right: Intermediate step while reflecting the first column over the $x = y$ line. Note that we start the column with distance five to the bounding box. This is needed to build the row upwards without hitting P if w is close to h.



(b) Left: Intermediate step while building the first reflected row. Note that the left side of the bounding box has been extended to make more space. Right: Intermediate step after reflecting the first column and row (original column and row are marked in light gray). Numbers show the matching between original position and new position of particles.



(c) Left: First row and column have been moved down and left to make space for the next row and column. Right: Merging the first two columns and rows.

Fig. 15 Intermediate steps of reflecting a polyomino over the $x = y$ line

of the leftmost column of P, and t_2 inside the bounding box marking the topmost row of P (see Fig. 15a). Afterwards, we extend the bounding box at the left side and the bottom side by six units without moving t_1, t_2 or t_c. This gives us a width of seven units between the polyomino and the bounding box at the left and bottom side (see Fig. 15a). Now we can move the first column rotated in clockwise direction five units below P (which is two units above the bounding box), as follows.

To find the next pixel whose state we want to copy, we move along the bounding box until we find t_c. Then we move down until we reach t_2, i.e., the robot finds a particle to the west. After storing the state s_p of the pixel p and removing its particle (if it exists), we move t_2 one position down. (If t_2 reached t_1, i.e., we would place t_2 on t_1, then we know that we have finished the column and proceed with the row by moving t_2 to the right of t_1 in the next iteration.) To place a particle denoting s_p, we move over t_1 to the right side of the bounding box and three units down and then six units to the left. From there, we can follow bridge particles and particles from P, until we reach a free position. We place a particle on this position if p was empty, or a particle below if p was occupied. In case we copy a row-pixel, we do a turn upwards and follow bridges and particles of P again. See Fig. 15. Note that we cannot place the row directly on top of the column or else we may get conflicts with bridges. We therefore build the row one unit to the left and above the column. Also note that during construction of the first column we may hit the left side of the bounding box. In this case we extend this side by one unit (see Fig. 15b).

After constructing a column and a row, we move the constructed row one unit to the left and two units down, we move the column one unit down and two units left, and delete the bridge particles on the fly. This gives us enough space to construct the next column and row in the same way (see Fig. 15c). Then we move t_1 one unit upwards and one unit to the right, we move t_2 four units below t_c (i.e., to the topmost row of P), and t_c one unit to the right.

When all columns and rows are constructed, we obtain a polyomino that is a reflected version of our desired polyomino. It is left to reflect vertically to obtain a polyomino rotated in counter-clockwise direction, or horizontally to obtain a polyomino that is rotated in clockwise direction. This can be done with the strategy described in Sect. 6.2.

Theorem 8 *There is a strategy to rotate a polyomino P by $\pm\frac{\pi}{2}$ within $\mathcal{O}((w+h)wh)$ unit steps, using $\mathcal{O}(w+h+|w-h|h)$ of additional space in $\mathcal{O}(|w-h|+1)$ extra rows and columns and $\mathcal{O}(w+h)$ auxiliary particles.*

Proof Like in our other algorithms, the number of steps to copy the state of a pixel to the desired position is bounded by $\mathcal{O}(w+h)$. Shifting the constructed row and column also takes the same number of steps. Therefore, constructing the reflected version of our desired polyomino needs $\mathcal{O}((w+h)wh)$ unit steps. Additionally we may have to extend one side of the bounding box. This can happen at most $\mathcal{O}(|w-h|)$ times, with each event needing $\mathcal{O}(h)$ unit steps. Because $\mathcal{O}(|w-h|)$ can be bounded by $\mathcal{O}(w+h)$, this does not change the total time complexity.

Because the width of the working space increases by $\mathcal{O}(|w-h|)$ and the height only increases by $\mathcal{O}(1)$, we need $\mathcal{O}(|w-h|h)$ of additional space. For the number of auxiliary particles, consider the number of particles used for the bounding box, for bridges, and marker particles. The bounding box is made from

$O(w + h + |w - h|) = O(w + h)$ particles, and there are only $O(w + h)$ bridge particles. As there are only $O(1)$ marker particles, we need in total $O(w + h)$ auxiliary particles. □

6.4 Scaling a Polyomino

Scaling a polyomino P by a factor c replaces each particle by a $c \times c$ square. This can easily be handled by our robot.

Theorem 9 *Given a constant c, the robot can scale the polyomino by c within $\mathcal{O}((w^2 + h^2)c^2N)$ unit steps using $\mathcal{O}(c^2wh)$ of additional space in $\mathcal{O}(c(w + h))$ additional rows and columns, using $\mathcal{O}(c^2N)$ auxiliary particles.*

Proof After constructing the bounding box, we place a particle denoting the current column. Suppose we are in the i-th column C_i. Then we shift all columns that lie to the right of C_i by c units to the right with cost of $\mathcal{O}((w - i) \cdot \sum_{j=i+1}^{h} cN_{C_j}) \subseteq \mathcal{O}(wcN)$, where N_{C_j} is the number of particles in the jth column. Because c is a constant, we can always find the next column to shift. Afterwards, we copy C_i exactly $c - 1$ times to the right of C_i, which costs $\mathcal{O}(N_{C_i}c)$ unit steps. Thus, extending a single column costs $\mathcal{O}(c \cdot (wN + N_{C_i}))$ and hence extending all columns costs $\mathcal{O}(cw^2N)$ unit steps in total.

Extending each row is done analogously. However, because each particle has already been copied c times, we obtain a running time of $\mathcal{O}(c^2h^2N)$. This implies a total running time of $\mathcal{O}((w^2 + h^2)c^2N)$. The proof for the particle and space complexity is straightforward. □

7 Conclusion

We have given a number of tools and functions for manipulating an arrangement of particles by a single robotic automaton with constant memory.

In our work, we focused on arrangements of square-shaped particles; it seems straightforward to apply our ideas to other arrangements, such as hexagonal particles. There are various further challenges, including more complex operations, explicitly dealing with the logistics of obtaining and moving new tiles from depots to their final destinations, as well as methods for sharing the work between multiple robots. Another extension is to consider three-dimensional arrangements of voxels, with robots restricted to motions over the surface, or being able to move through the interior. An additional set of problems arises by requiring the set of tiles and the robot location to stay connected; this is motivated by scenarios in which disconnected pieces may drift apart, e.g., in space. These and other questions are left for future work.

Acknowledgements A preliminary extended abstract appears in the Proceedings of the 13th Workshop on Algorithmic Foundations of Robotics (WAFR), 2018 [16]. Phillip Keldenich was supported by the DFG Research Unit "Controlling Concurrent Change", funding number FOR 1800, project FE407/17-2, "Conflict Resolution and Optimization".

Funding Open Access funding provided by Projekt DEAL.

References

1. Abdel-Rahman, A., Becker, A.T., Biediger, D.E., Cheung, K.C., Fekete, S.P., Gershenfeld, N.A., Hugo, S., Jenett, B., Keldenich, P., Niehs, E., Rieck, C., Schmidt, A., Scheffer, C., Yannuzzi, M.: Space ants: constructing and reconfiguring large-scale structures with finite automata. In: Symposium on Computations Geometry (SoCG), pp. 73:1–73:7 (2020). Video at https://www.ibr.cs.tu-bs.de/users/fekete/Videos/SoCG/2020/Space_final.mp4
2. Becker, A.T., Demaine, E.D., Fekete, S.P., Lonsford, J., Morris-Wright, R.: Particle computation: complexity, algorithms, and logic. Nat. Comput. **18**(1), 181–201 (2019)
3. Blum, M., Kozen, D.: On the power of the compass (or, why mazes are easier to search than graphs). In: Symposium on Foundations of Computer Science (FOCS), pp. 132–142 (1978)
4. Bonato, A., Nowakowski, R.J.: The Game of Cops and Robbers on Graphs. AMS, Providence (2011)
5. Das, S.: Mobile agents in distributed computing: network exploration. Bull. Eur. Assoc. Theor. Comput. Sci. **109**, 54–69 (2013)
6. Daymude, J.J., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: Improved leader election for self-organizing programmable matter. In: International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS), pp. 127–140 (2017)
7. Demaine, E., Demaine, M., Hoffmann, M., O'Rourke, J.: Pushing blocks is hard. Comput. Geom. **26**(1), 21–36 (2003)
8. Demaine, E.D., Fekete, S.P., Scheffer, C., Schmidt, A.: New geometric algorithms for fully connected staged self-assembly. Theoret. Comput. Sci. **671**, 4–18 (2017)
9. Derakhshandeh, Z., Dolev, S., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: Brief announcement: Amoebot—a new model for programmable matter. In: ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 220–222 (2014)
10. Derakhshandeh, Z., Gmyr, R., Porter, A., Richa, A.W., Scheideler, C., Strothmann, T.: On the runtime of universal coating for programmable matter. In: International Conference on DNA Computing and Molecular Programming (DNA), pp. 148–164 (2016)
11. Derakhshandeh, Z., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: An algorithmic framework for shape formation problems in self-organizing particle systems. In: International Conference on Nanoscale Computing and Communication (NANOCOM), pp. 21 (2015)
12. Derakhshandeh, Z., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: Universal shape formation for programmable matter. In: ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 289–299 (2016)
13. Derakhshandeh, Z., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: Universal coating for programmable matter. Theoret. Comput. Sci. **671**, 56–68 (2017)

14. Derakhshandeh, Z., Gmyr, R., Strothmann, T., Bazzi, R., Richa, A.W., Scheideler, C.: Leader election and shape formation with self-organizing programmable matter. In: International Conference on DNA Computing and Molecular Programming (DNA), pp. 117–132 (2015)
15. Di Luna, G.A., Flocchini, P., Santoro, N., Viglietta, G., Yamauchi, Y.: Shape formation by programmable particles. Distrib. Comput. **33**, 1–33 (2019)
16. Fekete, S.P., Gmyr, R., Hugo, S., Keldenich, P., Scheffer, C., Schmidt, A.: CADbots: algorithmic aspects of manipulating porgrammable matter with finite automata. In: Algorithmic Foundations of Robotics XIII (WAFR 2018), Springer Proceedings in Advanced Robotics, pp. 725–743 (2020)
17. Fomin, F.V., Thilikos, D.M.: An annotated bibliography on guaranteed graph searching. Theoret. Comput. Sci. **399**(3), 236–245 (2008)
18. Gmyr, R., Hinnenthal, K., Kostitsyna, I., Kuhn, F., Rudolph, D., Scheideler, C.: Shape recognition by a finite automaton robot. In: International Symposium on Mathematical Foundations of Computer Science (MFCS), pp. 52:1–52:15 (2018)
19. Gmyr, R., Hinnenthal, K., Kostitsyna, I., Kuhn, F., Rudolph, D., Scheideler, C., Strothmann, T.: Forming tile shapes with simple robots. In: International Conference on DNA Computing and Molecular Programming (DNA), pp. 122–138 (2018)
20. Hurtado, F., Molina, E., Ramaswami, S., Sacristán, V.: Distributed reconfiguraiton of 2D lattice-based modular robotic systems. Auton. Robots **38**(4), 383–413 (2015)
21. Lund, K., Manzo, A., Dabby, N., Michelotti, N., Johnson-Buck, A., Nangreave, J., Taylor, S., Pei, R., Stojanovic, M., Walter, N., Winfree, E.: Molecular robots guided by prescriptive landscapes. Nature **465**(7295), 206–210 (2010)
22. Markou, E.: Identifying hostile nodes in networks using mobile agents. Bull. Eur. Assoc. Theor. Comput. Sci. **108**, 93–129 (2012)
23. Michail, O., Spirakis, P.G.: Simple and efficient local codes for distributed stable network construction. Distrib. Comput. **29**(3), 207–237 (2016)
24. Niehs, E., Schmidt, A., Scheffer, C., Biediger, D.E., Yannuzzi, M., Jenett, B., Abdel-Rahman, A., Cheung, K.C., Becker, A.T., Fekete, S.P.: Recognition and reconfiguration of lattice-based cellular structures by simple robots. In: International Conference on Robotics and Automation (ICRA) (2020) **(to appear)**
25. Omabegho, T., Sha, R., Seeman, N.: A bipedal DNA Brownian motor with coordinated legs. Science **324**(5923), 67–71 (2009)
26. Patitz, M.J.: An introduction to tile-based self-assembly and a survey of recent results. Nat. Comput. **13**(2), 195–224 (2014)
27. Pelc, A.: Deterministic rendezvous in networks: a comprehensive survey. Networks **59**(3), 331–347 (2012)
28. Reif, J.H., Sahu, S.: Autonomous programmable DNA nanorobotic devices using dnazymes. Theoret. Comput. Sci. **410**, 1428–1439 (2009)
29. Shah, A.N.: Pebble automata on arrays. Comput. Graph. Image Process. **3**(3), 236–246 (1974)
30. Shin, J., Pierce, N.: A synthetic DNA walker for molecular transport. J. Am. Chem. Soc. **126**, 4903–4911 (2004)
31. Terada, Y., Murata, S.: Automatic modular assembly system and its distributed control. Int. J. Robot. Res. **27**(3–4), 445–462 (2008)
32. Thubagere, A., Li, W., Johnson, R., Chen, Z., Doroudi, S., Lee, Y., Izatt, G., Wittman, S., Srinivas, N., Woods, D., Winfree, E., Qian, L.: A cargo-sorting DNA robot. Science **357**(6356), eaan6558 (2017)
33. Wang, Z., Elbaz, J., Willner, I.: A dynamically programmed DNA transporter. Angew. Chem. Int. Ed. **51**(48), 4322–4326 (2012)
34. Wickham, S., Bath, J., Katsuda, Y., Endo, M., Hidaka, K., Sugiyama, H., Turberfield, A.: A DNA-based molecular motor that can navigate a network of tracks. Nat. Nanotechnol. **7**(3), 169–173 (2012)
35. Woods, D., Chen, H., Goodfriend, S., Dabby, N., Winfree, E., Yin, P.: Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In: Innovations in Theoretical Computer Science (ITCS), pp. 353–354 (2013)