# Shape Formation by Programmable Particles*

**Giuseppe A. Di Luna[1], Paola Flocchini[2], Nicola Santoro[3], Giovanni Viglietta[4], and Yukiko Yamauchi[5]**

1 **University of Ottawa, Ottawa, Canada**
gdiluna@uottawa.ca
2 **University of Ottawa, Ottawa, Canada**
paola.flocchini@uottawa.ca
3 **Carleton University, Ottawa, Canada**
santoro@scs.carleton.ca
4 **University of Ottawa, Ottawa, Canada**
gvigliet@uottawa.ca
5 **Kyushu University, Fukuoka, Japan**
yamauchi@inf.kyushu-u.ac.jp

## Abstract

Shape formation (or pattern formation) is a basic distributed problem for systems of computational mobile entities. Intensively studied for systems of autonomous mobile robots, it has recently been investigated in the realm of programmable matter, where entities are assumed to be small and with severely limited capabilities. Namely, it has been studied in the geometric Amoebot model, where the anonymous entities, called particles, operate on a hexagonal tessellation of the plane and have limited computational power (they have constant memory), strictly local interaction and communication capabilities (only with particles in neighboring nodes of the grid), and limited motorial capabilities (from a grid node to an empty neighboring node); their activation is controlled by an adversarial scheduler. Recent investigations have shown how, starting from a well-structured configuration in which the particles form a (not necessarily complete) triangle, the particles can form a large class of shapes. This result has been established under several assumptions: agreement on the clockwise direction (i.e., chirality), a sequential activation schedule, and randomization (i.e., particles can flip coins to elect a leader).

In this paper we provide a characterization of which shapes can be formed deterministically starting from any simply connected initial configuration of $n$ particles. The characterization is constructive: we provide a universal shape formation algorithm that, for each feasible pair of shapes $(S_0, S_F)$, allows the particles to form the final shape $S_F$ (given in input) starting from the initial shape $S_0$, unknown to the particles. The final configuration will be an appropriate scaled-up copy of $S_F$ depending on $n$.

If randomization is allowed, then any input shape can be formed from any initial (simply connected) shape by our algorithm, provided that there are enough particles.

Our algorithm works without chirality, proving that chirality is computationally irrelevant for shape formation. Furthermore, it works under a strong adversarial scheduler, not necessarily sequential.

We also consider the complexity of shape formation both in terms of the number of rounds and the total number of moves performed by the particles executing a universal shape formation algorithm. We prove that our solution has a complexity of $O(n^2)$ rounds and moves: this number of moves is also asymptotically worst-case optimal.

**1998 ACM Subject Classification** F.1.1 Models of Computation, F.2.2 Nonnumerical Algorithms and Problems, I.2.11 Distributed Artificial Intelligence, I.2.9 Robotics

* Full version available at https://arxiv.org/abs/1705.03538. A short version appeared as "Brief Announcement: Shape Formation by Programmable Particles" in Proceedings of DISC 2017.



21st International Conference on Principles of Distributed Systems (OPODIS 2017).
Editors: James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão; Article No. 31; pp. 31:1–31:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**Keywords and phrases** Shape formation, pattern formation, programmable matter, Amoebots, leader election, distributed algorithms

**Digital Object Identifier** 10.4230/LIPIcs.OPODIS.2017.31

# 1 Introduction

## 1.1 Background

The term *programmable matter*, introduced by Toffoli and Margolus over a quarter century ago [23], is used to denote matter that has the ability to change its physical properties (e.g., shape, color, etc.) in a programmable fashion, based upon user input or autonomous sensing. Often programmable matter is envisioned as a very large number of very small locally interacting computational particles, programmed to collectively perform a complex task. Such particles could have applications in a variety of important situations: smart materials, autonomous monitoring and repair, minimal invasive surgery, etc.

Several theoretical models for programmable matter have been proposed, ranging from DNA self-assembly systems (e.g., [18, 19, 21]) to metamorphic robots (e.g., [3, 16, 24]) to nature-inspired synthetic insects and micro-organisms (e.g., [10, 11]).

Of particular interest, from the distributed computing viewpoint, is the *geometric Amoebot* model [2, 5, 6, 7, 8, 9, 10, 14]. In this model, introduced in [10] and so called because inspired by the behavior of amoeba, programmable matter is viewed as a swarm of decentralized autonomous self-organizing entities, operating on a hexagonal tessellation of the plane. These entities, called *particles*, are constrained by having simple computational capabilities (they are finite-state machines), strictly local interaction and communication capabilities (only with particles located in neighboring nodes of the hexagonal grid), and limited motorial capabilities (from a grid node to an empty neighboring node); furthermore, their activation is controlled by an adversarial (but fair) synchronous scheduler. A feature of the Amoebot model is that particles can be in two modes: *contracted* and *expanded*. When contracted, a particle occupies only one node, while when expanded the particle occupies two neighboring nodes; it is indeed this ability of a particle to expand and contract that allows it to move on the grid.

In the Amoebot model, the research focus has been on applications such as coating [5, 9], gathering [2], and shape formation [7, 8, 10]. The latter is also the topic of our investigation.

The *shape formation* problem is prototypical for systems of self-organizing entities. This problem, called *pattern formation* in swarm robotics, requires the entities to move in the spatial universe they inhabit in such a way that, within finite time, their positions form the geometric shape given in input (modulo translation, rotation, scaling, and reflection), and no further changes occur. Indeed, this problem has been intensively studied in active systems such as autonomous mobile robots (e.g., [1, 4, 12, 13, 22]) and modular robotic systems (e.g., [17, 20]).

Shape formation for Amoebots has been investigated in [7, 8, 10], taking into account that, due to the ability of particles to expand, it might be possible to form shapes whose size is larger than the number of particles.

The pioneering study of [7] on *shape formation* in the geometric Amoebot model showed how particles can build simple shapes, such as a hexagon or a triangle. Subsequent investigations [8] have recently shown how, starting from a well-structured configuration where the particles form a (not necessarily complete) triangle, they can form a larger class of shapes under several assumptions, including randomization (which is used to elect a leader), chirality

(i.e., a globally consistent circular orientation of the plane shared by all particles), and a sequential activation schedule (i.e., at each time unit the scheduler selects only one particle which will interact with its neighbors and possibly move).

These results and assumptions immediately and naturally open fundamental research questions, including: Are other shapes formable? What can be done deterministically? Is chirality necessary? What happens if the scheduler is not sequential? What if the initial configuration is not well structured? Notice that, without the availability of a unique leader (provided by randomization), dropping the chirality assumption becomes a problem with a non-sequential schedule.

In this paper, motivated and stimulated by these questions, we continue the investigation on shape formation in the geometric Amoebot model and provide some definitive answers.

## 1.2 Main Contributions

We continue the investigation, significantly extending the existing results. Among other things, we provide a constructive characterization of which shapes $S_F$ can be formed *deterministically* starting from an unknown *simply connected* initial configuration $S_0$ of $n$ particles (i.e., a connected configuration without "holes").

As in [8], we assume that the size of the description of $S_F$ is constant with respect to the size of the system, so that it can be encoded by each particle as part of its internal memory. Such a description is available to all the particles at the beginning of the execution, and we call it their "input". The particles will form a final configuration that is an appropriate scaling, translation, rotation, and perhaps reflection of the input shape $S_F$. Since all particles of $S_0$ must be used to construct $S_F$, the scale $\lambda$ of the final configuration depends on $n$: we stress that $\lambda$ is unknown to particles, and they must determine it autonomously.

Given two shapes $S_0$ and $S_F$, we say that the pair $(S_0, S_F)$ is *feasible* if there exists a deterministic algorithm that, in every execution and regardless of the activation schedule, allows the particles to form $S_F$ starting from $S_0$ and no longer move.

On the contrary, a pair $(S_0, S_F)$ of shapes is *unfeasible* when the symmetry of the initial configuration $S_0$ prevents the formation of the final shape $S_F$. In Section 2, we formalize the notion of *unbreakable* symmetry of shapes embedded in triangular grids, and in Theorem 1 we show that starting from an unbreakable $k$-symmetric configuration only unbreakable $k$-symmetric shapes can be formed.

For all the feasible pairs, we provide a *universal shape formation algorithm* in Section 3. This algorithm does not need any information on $S_0$, except that it is simply connected.

These results concern the *deterministic* formation of shapes. As a matter of fact, our algorithm uses a deterministic leader election algorithm as a subroutine (Sections 3.1–3.4). If the initial shape $S_0$ is unbreakably $k$-symmetric, such an algorithm may elect as many as $k$ neighboring leader particles, where $k \in \{1, 2, 3\}$. It is trivial to see that, with a constant number of coin tosses, we can elect a unique leader among these $k$ with arbitrarily high probability. Thus, our results immediately imply the existence of a *randomized* universal shape formation algorithm for *any* pair of shapes $(S_0, S_F)$ where $S_0$ is simply connected. This extends the result of [8], which assumes the initial configuration to be a (possibly incomplete) triangle.

Additionally, our notion of shape generalizes the one used in [8], where a shape is only a collection of triangles, while we include also 1-dimensional segments as its constituting elements. In Section 4 we will show how the concept of shape can be further generalized to include essentially anything that is Turing-computable.

Our algorithm works under a stronger adversarial scheduler that activates an arbitrary number of particles at each stage (i.e., not necessarily just one, like the sequential scheduler), and with a slightly less demanding communication system.

Moreover, in our algorithm no chirality is assumed: indeed, unlike in [8], different particles may have different handedness. On the contrary, in the examples of unfeasibility given in Theorem 1, all particles have the same handedness. Together, these two facts allows us to conclude that *chirality is computationally irrelevant* for shape formation.

Finally, we analyze the complexity of shape formation in terms of the total number of *moves* (i.e., contractions and expansions) performed by $n$ particles executing a universal shape formation algorithm, as well as in terms of the total number of *rounds* (i.e., spans of time in which each particle is activated at least once, also called *epochs*) taken by the particles. We first prove that any universal shape formation algorithm requires $\Omega(n^2)$ moves in some cases (Theorem 2). We then show that the total number of moves of our algorithm is $O(n^2)$ in all cases (Theorem 3): that is, our solution is asymptotically worst-case optimal. The time complexity of our algorithm is also $O(n^2)$ rounds, and optimizing it is left as an open problem (we can reduce it to $O(n \log n)$, and we have a lower bound of $\Omega(n)$).

Obviously, we must assume the size of $S_0$ (i.e., the number of particles that constitute it) to be sufficiently large with respect to the input description of the final shape $S_F$. More precisely, denoting the size of $S_F$ as $m$, we assume $n$ to be lower-bounded by a cubic function of $m$. A similar restriction is also found in [8].

To the best of our knowledge, all the techniques employed by our universal shape formation algorithm are new.

Further technical details and most proofs can be found in the full version of the paper [15].

## 2 Model and Preliminaries

**Particles.** A *particle* is a conceptual model for a computational entity that lives in an infinite regular triangular grid $G$, which we imagine as embedded in the Euclidean plane $\mathbb{R}^2$. A particle may occupy either one vertex of $G$ or two adjacent vertices: in the first case, the particle is said to be *contracted*; otherwise, it is *expanded*. When it is expanded, one of the vertices it occupies is called its *head*, and the other vertex is its *tail*. A particle may move through $G$ by repeatedly expanding toward a neighboring vertex of $G$ and contracting into its head.[1]

No vertex of $G$ can ever be occupied by more than one particle at the same time. Accordingly, a contracted particle cannot expand toward a vertex that is already occupied by another particle. If two or more particles attempt to expand toward the same (unoccupied) vertex at the same time, only one of them succeeds, chosen arbitrarily by an adversarial scheduler (see below).

In our model, time is "discrete", i.e., it is an infinite ordered sequence of instants, called *stages*. Say that in the graph $G$ there is a set $P$ of particles, which we call a *system*. At each stage, some particles of $P$ are *active*, and the others are *inactive*. We may think of the activation of a particle as an act of an adversarial *scheduler*, which arbitrarily and unpredictably decides which particles are active at each stage. The only restriction on the scheduler is a bland *fairness* constraint, requiring that each particle be active for infinitely many stages in total. That is, the scheduler can never keep a particle inactive forever.

[1] The model in [8] allows a special type of coordinated move called "handover". Since we will not need our particles to perform this type of move, we omit it from our model.

When a particle is activated for a certain stage, it "looks" at the vertices of $G$ adjacent to its head, discovering if they are currently unoccupied, or if they are head or tail vertices of some particle. All particles are indistinguishable (i.e., they are *anonymous*). Each active particle may then decide to either expand, contract, or stay still for that stage. All these operations are performed by all active particles simultaneously, and take exactly one stage. So, when the next stage starts, a new set of active particles is selected, which observe their surroundings and move, and so on.

Each particle has an *internal state* that it can modify every time it is activated. The internal state of any particle must be picked from a finite set: particles have an amount of "memory" that is constant with respect to the size of the system, $n$.

Two particles can also *communicate* by sending each other *messages* taken from a finite set, provided that their heads are adjacent vertices of $G$. A particle reads the incoming messages from all its neighbors as soon as it is activated. If a second message is sent through an edge of $G$ before the first one is read, the first message is overwritten and becomes inaccessible.

Each particle labels the six edges incident to each vertex of $G$ with *port numbers*, going from 0 to 5. Each particle uses a consistent numbering that is invariant under translation on $G$. However, different particles may disagree on which of the edges incident to a vertex has port number 0 and whether the numbering should follow the clockwise or counterclockwise order: this is called the particles' *handedness*. So, the handedness of a particle does not change as the particle moves, but different particles may have different handedness.

At each stage, each active particle looks at its surroundings to see which neighboring vertices are occupied, and it reads the incoming messages. Based on these and on its internal state, the particle executes a *deterministic algorithm* that computes a new internal state, the messages to be sent to the neighbors, and whether the particle should expand to some adjacent vertex, contract, or stay still.

We assume that, when stage 0 starts, all particles are contracted, they all have the same predefined internal state, and there are no messages pending between particles.

**Shape formation.** A *shape* is a non-empty connected set consisting of the union of finitely many edges and faces of $G$. We stress that a shape is not a subgraph of an abstract graph, but it is a subset of $\mathbb{R}^2$, i.e., a geometric set. A shape $S$ is *simply connected* if the set $\mathbb{R}^2 \setminus S$ is connected (intuitively, $S$ has no "holes"). The *size* of a shape is the number of vertices of $G$ that lie in it.

We say that two shapes $S$ and $S'$ are *equivalent* if $S'$ is obtained from $S$ by a similarity transformation, i.e., a composition of a translation, a rotation, an isotropic scaling by a positive factor, and an optional reflection. A shape $S$ is *minimal* if no shape that is equivalent to it has a smaller size. Let $\sigma$ be a similarity transformation such that $S' = \sigma(S)$, where $S$ is minimal: we say that the (positive) scale factor of $\sigma$ is the *scale* of $S'$.

A system of particles in $G$ *forms* a shape $S$ if the vertices of $G$ that are occupied by particles are exactly the ones that lie in $S$. An $(S_0, S_F)$*-shape formation algorithm* is an algorithm that makes a system of particles form a shape *equivalent* to $S_F$ (not necessarily $S_F$ itself), provided that they form shape $S_0$ at stage 0. That is, however the port labels of each particle are arranged, and whatever the choices of the scheduler are. After forming the shape, the particles should no longer move. If such an algorithm exists, $(S_0, S_F)$ is called a *feasible* pair of shapes.

In the rest of this paper, we will characterize the feasible pairs of shapes $(S_0, S_F)$, provided that $S_0$ is simply connected and its size is not too small. That is, for every such pair of

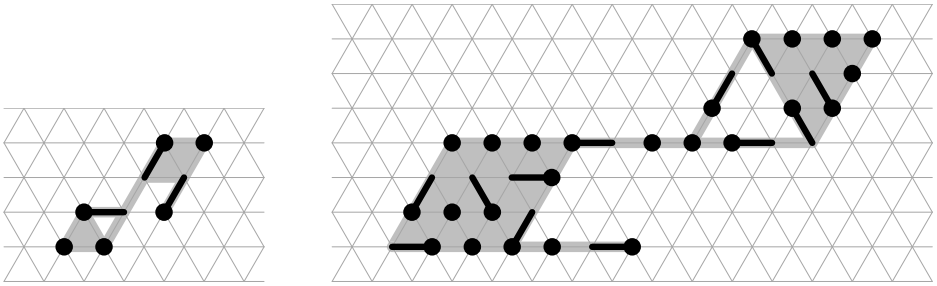


**Figure 1** Two systems of particles forming equivalent shapes. Contracted particles are represented as black dots; expanded particles are black segments (a dot represents a particle's head). Shapes are indicated by gray blobs. The shape on the left is minimal; the one on the right has scale 3.

shapes, we will either prove that no shape formation algorithm exists (Theorem 1), or we will give an explicit shape formation algorithm. We will actually give a *universal shape formation algorithm* (Theorem 3), which only takes the "final shape" $S_F$ (or a representation thereof) as a parameter, and has no information on the "initial shape" $S_0$, except that it is simply connected. As in [8], we assume that the size of the parameter $S_F$ is constant with respect to the size of the system, so that $S_F$ can be encoded by each particle as part of its internal memory.

**Unformable shapes.** A shape is said to be *unbreakably $k$-symmetric*, for some integer $k > 1$, if it has a center of $k$-fold rotational symmetry that does not coincide with any vertex of $G$. Observe that there exist unbreakably $k$-symmetric shapes only for $k = 2$ and $k = 3$.

If the system initially forms an unbreakably $k$-symmetric shape, the port labels of symmetric particles happen to be symmetric, and the scheduler always activates symmetric particles simultaneously, then the system never ceases to form an unbreakably $k$-symmetric shape. Therefore, we have the following:

▶ **Theorem 1.** *If $(S_0, S_F)$ is a feasible pair and $S_0$ is unbreakably $k$-symmetric, then any minimal shape that is equivalent to $S_F$ is also unbreakably $k$-symmetric.*

In Section 3, we are going to prove that the condition of Theorem 1 characterizes the feasible pairs of shapes, provided that $S_0$ is simply connected, and the size of $S_0$ is large enough with respect to the size of $S_F$.

**Measuring movements and rounds.** We are also concerned to measure the total number of *moves* performed by a system of size $n$ executing a shape formation algorithm. For instance, if the particles initially form a (full) regular hexagon, they must make at least $\Omega(n^2)$ moves in total to form a line segment (because $\Omega(n)$ particles must move over a distance of $\Omega(n)$). Hence we have the following lower bound:

▶ **Theorem 2.** *A system of $n$ particles executing any universal shape formation algorithm performs $\Omega(n^2)$ moves in total.*

In Section 3, we will prove that our universal shape formation algorithm requires $O(n^2)$ moves in total, and is therefore worst-case optimal with respect to this parameter.

Similarly, we want to measure how many *rounds* it takes the system to form the final shape (a round is a span of time in which each particle is activated at least once). We will show that our universal shape formation algorithm takes $O(n^2)$ rounds.

## 3 Universal Shape Formation Algorithm

**Algorithm structure.** The universal shape formation algorithm takes a "final shape" $S_F$ as a parameter: this is encoded in the initial states of all particles. Without loss of generality, we will assume $S_F$ to be minimal. The algorithm consists of seven phases:

1. A *lattice consumption phase*, in which the initial shape $S_0$ is "eroded" until 1, 2, or 3 pairwise adjacent particles are identified as "candidate leaders". No particle moves in this phase: only messages are exchanged. This phase ends in $O(n)$ rounds.
2. A *spanning forest construction phase*, in which a spanning forest of $S_0$ is constructed, where each candidate leader is the root of a tree. No particle moves, and the phase ends in $O(n)$ rounds.
3. A *handedness agreement phase*, in which all particles assume the same handedness as the candidate leaders (some candidate leaders may be eliminated in the process). In this phase, at most $O(n)$ moves are made. However, at the end, the system forms $S_0$ again. This phase ends in $O(n)$ rounds.
4. A *leader election phase*, in which the candidate leaders attempt to break symmetries and elect a unique leader. If they fail to do so, and $k > 1$ leaders are left at the end of this phase, it means that $S_0$ is unbreakably $k$-symmetric, and therefore the "final shape" $S_F$ must also be unbreakably $k$-symmetric (cf. Theorem 1). No particle moves, and the phase ends in $O(n^2)$ rounds.
5. A *straightening phase*, in which each leader coordinates a group of particles in the formation of a straight line. The $k$ resulting lines have the same length. At most $O(n^2)$ moves are made, and the phase ends in $O(n^2)$ rounds.
6. A *role assignment phase*, in which the particles determine the scale of the shape $S'_F$ (equivalent to $S_F$) that they are actually going to form. Each particle is assigned an identifier that will determine its behavior during the formation process. No particle moves, and the phase ends in $O(n^2)$ rounds.
7. A *shape composition phase*, in which each straight line of particles, guided by a leader, is reconfigured to form an equal portion of $S'_F$. At most $O(n^2)$ moves are made, and the phase ends in $O(n^2)$ rounds.

No a-priori knowledge of $S_0$ is needed to execute this algorithm ($S_0$ just has to be simply connected), while $S_F$ must of course be known to the particles and have constant size, so that its description can reside in their memory. Note that the knowledge of $S_F$ is needed only in the last two phases of the algorithm.

**Synchronization.** As long as there is a unique (candidate) leader in the system, there are no synchronization problems: this one particle coordinates all others, and autonomously decides when each phase ends and the next phase starts.

However, if there are $k > 1$ (candidate) leaders, there are possible issues arising from the intrinsic asynchronicity of our particle model. Typically, a (candidate) leader will be in charge of coordinating only a portion of the system, and we want to avoid the undesirable situation in which different leaders are executing different phases of the algorithm.

So, a basic synchronization protocol is executed "in parallel" with the shape formation algorithm. This protocol simply makes sure that, whenever the (candidate) leaders are neighbors (which is true most of the time), they exchange messages containing the identifiers of the phases that they are currently executing, and those who are ahead wait until the others have caught up (see [15] for more details).

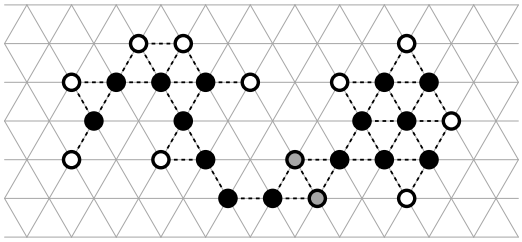


**Figure 2** The particles in white or gray are corner particles; the two in gray are locked particles. Dashed lines indicate adjacencies between particles.

## 3.1 Lattice Consumption Phase

The goal of this phase is to identify 1, 2, or 3 *candidate leaders*. This is done without making any movements, but only exchanging messages. Each particle's internal state has a flag (i.e., a bit) called *Eligible*. All particles start the execution in the same state, with the Eligible flag set. As the execution proceeds, eligible particles will gradually "eliminate themselves" by clearing their Eligible flag. This is achieved through a process similar to erosion, which starts from the boundary of the initial shape and proceeds toward its interior.

The "consumption" of the initial shape $S_0$ starts from its *corner particles*: roughly speaking, these are the particles located at convex vertices of $S_0$ or at the end of dangling edges, as Figure 2 shows.

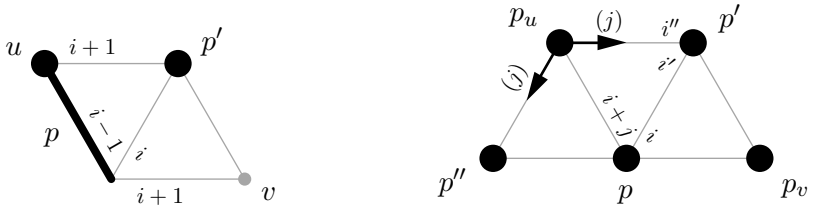
By looking at its surroundings, a particle knows if it is a corner particle or not; then, it can communicate this information to its neighbors. In most cases, if $S_0$ is a simply connected shape, removing any subset of its corner particles leaves the shape simply connected (i.e., it does not disconnect it or create holes in it). There is just one exception, represented by the two gray particles in Figure 2: removing both particles disconnects the shape. Fortunately, such a configuration is unique and can be locally recognized by the gray particles (provided that they send each other a description of their neighborhoods), which identify themselves as *locked particles*. A locked particle waits and remains eligible until some of its neighbors have eliminated themselves.

We can also prove that any simply connected shape contains non-locked corner particles, and therefore the erosion process eventually succeeds: when only 1, 2, or 3 pairwise adjacent eligible particles remain, they are the candidate leaders (technically, they become candidate leaders by setting an internal *Candidate* flag). Note that taking another erosion step from there may eliminate all particles (since they may behave symmetrically), and therefore we must stop at this point and use other methods to elect a leader.

## 3.2 Spanning Forest Construction Phase

The spanning forest construction phase starts when 1, 2, or 3 pairwise adjacent candidate leaders have been identified, and no other particle is eligible. In this phase, each candidate leader becomes the root of a tree embedded in $G$. Eventually, the set of these trees will be a spanning forest of the subgraph of $G$ induced by the system of particles.

The algorithm is straightforward: the trees are constructed starting from the candidate leaders, and the process involves more and more adjacent particles until all of them are included. Each particle remembers which port label corresponds to its parent and which ones correspond to its children. The details of this algorithm can be found in [15].




**(a)** A neighbor $u$ is unoccupied. **(b)** Both neighbors $u$ and $v$ are occupied.

**Figure 3** The subroutine by which $p$ communicates its handedness to $p'$. The labels on edges indicate port numbers. The labels in parentheses are messages.

## 3.3 Handedness Agreement Phase

When a candidate leader is notified by all its children that its tree can no longer be expanded, it transitions to the handedness agreement phase. Recall that each particle may label ports in clockwise or counterclockwise order: this is called the particle's handedness. By the end of this phase, all particles will agree on a common handedness. The agreement process starts at the candidate leaders and proceeds through the spanning forest constructed in the previous phase, from parents to children.

The core subroutine of this algorithm involves a generic particle $p$ that intends to communicate its handedness to one of its children $p'$ (in the tree constructed in the previous phase). Of course, this cannot be done simply by message passing, because a particle has no direct way to tell a neighbor which direction it "thinks" is clockwise: a special technique is required, which is summarized in Figure 3.

Let $u$ and $v$ be the two vertices of $G$ that are neighbors to both $p$ and $p'$. Suppose first that one of them is unoccupied, say $u$. Without loss of generality, $p$ sees $u$ to the left of $p'$, as in Figure 3a. Then, $p$ expands toward $u$ and sends a message to $p'$ saying "I-Moved-Left". If $p'$ gets this message from its right neighbor, it has the same handedness as $p$; otherwise, it inverts its own handedness to match $p$'s. Then, $p$ goes back to its original location.

Suppose now that both $u$ and $v$ are occupied by particles $p_u$ and $p_v$, as in Figure 3b. Then, $p$ sends a message to $p_u$ saying "You-are-my-Left-Neighbor" and one to $p_v$ saying "You-are-my-Right-Neighbor". Then, $p_u$ is supposed to message $p'$, but it does not know where it is, except that it neighbors both $p_u$ and $p$. As there are two such locations, $p_u$ sends an "I-am-the-Left-Neighbor" message to both. Again, if $p'$ receives this message from its right neighbor (with respect to its parent $p$), it knows it has the same handedness as $p$; otherwise, it inverts its handedness. In the meantime, $p_v$ does a similar thing, sending "I-am-the-Right-Neighbor" messages. $p'$ waits until it has received messages from both $p_u$ and $p_v$, and then it notifies its parent $p$ that the procedure is over.

This subroutine is executed between any non-leaf particle and all its children, starting from the candidate leaders. Before that, the candidate leaders execute a variation of this subroutine among themselves, to make sure they all have the same handedness (if they do not, then all but one are eliminated). At the end of this process, all particles have agreed on the same handedness.

The above algorithm has some obvious flaws, because several particles may be executing the subroutine at the same time, possibly interfering with each other if they try to expand toward the same vertex or if they send messages to the same particle. To solve the first problem, the particle $p$ that initiates the subroutine does not try to expand to $u$ if it

remembers that it was originally occupied by some other particle; instead, it waits for $p_u$ to come back or expands to $v$. Also, if two particles try to expand toward the same location, only one succeeds: the other particle waits and tries again later.

To solve the second problem, when both $u$ and $v$ are occupied, $p$ first *locks* both $p_u$ and $p_v$, then notifies $p'$, and then proceeds with the subroutine. When a particle is locked, it finishes the current subroutine before starting a new one with another neighbor. So, when $p_u$ sends its "I-am-the-Left-Neighbor" or "I-am-the-Right-Neighbor" to the wrong particle $p''$, this particle is able to realize that it is not the intended addressee, and ignores the message. This is because the parent of $p''$ cannot have locked $p_u$ and then notified $p''$, since $p_u$ has already been locked by $p$ for an operation with $p'$.

A consequence of this protocol is that, if $p$ can lock only $p_u$, say, because $p_v$ has already been locked by some other particle, then $p$ keeps $p_u$ locked while it waits for $p_v$ to become unlocked: note that this potentially gives rise to *deadlocks*. To prevent them, each particle executes the subroutine with only one child at a time; when all its children have the same handedness, then it "authorizes" its first child to proceed with its own children, etc. This way, at any time there can be at most one pair of particles involved in the subroutine in each tree of the spanning forest. Since there are at most three trees, it is then straightforward to prove that deadlocks are impossible (see [15] for the details).

## 3.4 Leader Election Phase

In this phase, one candidate leader is finally elected to be the unique leader, provided that the shape $S_0$ is not unbreakably $k$-symmetric. If it is, then the $k$ candidate leaders may be unable to decide who should be elected, and hence all of them become leaders.

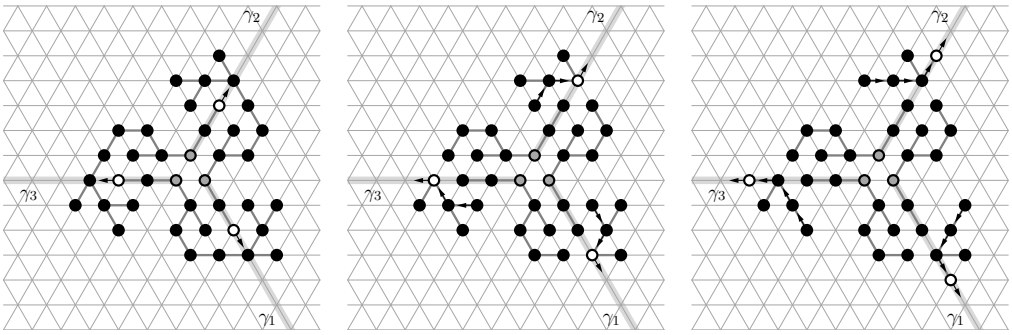
In order to elect a leader, the candidates "scan" their respective trees of the spanning forest, searching for asymmetric features of $S_0$ that would allow them to decide which candidate should become the leader. This task is made possible by the fact that all particles agree on the same handedness.

The algorithm is divided in steps: at each step, a particle $q$ in each tree sends a constant-length *code* to its parent, describing its neighborhood; the message is then forwarded all the way to the candidate leader at the root of the tree. Such a code contains information on all the neighbors of $q$, starting from the parent and proceeding in clockwise order: the information that is encoded essentially tells whether each neighbor is a child of $q$, or some other particle, or an unoccupied vertex of $G$.

Once all candidate leaders have obtained a code from a particle in their respective tree, they send each other these codes and compare them. If the codes are not all equal, the candidates are able to elect a unique leader. Otherwise, the "election attempt" fails, and each candidate leader asks one of its children for another neighborhood code.

In the first step, the candidate leaders compare their own neighborhood codes. If the election attempt fails, each of them asks its first child in clockwise order for its neighborhood code. As the attempts keep failing, the particles in each tree are queried as in a preorder traversal, where the children of every particle are always queried in clockwise order. Since all particles agree on the same handedness, the candidate leaders keep comparing the codes of symmetric particles in their respective trees, until asymmetric particles are found. In turn, these particles have the same handedness, and so they produce the same neighborhood codes only if their surroundings are indeed symmetric.

It follows that a unique leader is not elected only if $S_0$ is unbreakably $k$-symmetric. If all the election attempts fail, the $k$ candidate leaders have no choice but to become all leaders.




**(a)** Each pioneer is obstructed by a particle on its directrix.

**(b)** The obstructing particles detach from their parents and become the new pioneers.

**(c)** Each new pioneer pulls a chain of particles from the closest entry point.

**Figure 4** Three stages of the straightening phase. The particles in gray are the leaders; the ones in white are the pioneers. The edges of the spanning forest are drawn in dark gray, and the arrows indicate where the particles are directed in the pulling procedure.

## 3.5 Straightening Phase

At the beginning of this phase, there are $k = 1$, $k = 2$, or $k = 3$ leaders, each of which is the root of a tree of particles. These $k$ trees are rotated copies of each other, and the leaders are pairwise adjacent. The goal of this phase is to arrange the particles in a way that will make the final phase of the algorithm simpler to design: that is, $k$ straight lines radiating from the center of the figure.

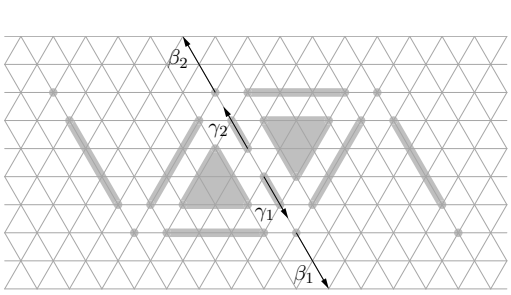
So, in this phase, each leader $p_i$ coordinates the "straightening" of its tree. $p_i$ chooses a ray in the plane (i.e., a half-line) as its *directrix* $\gamma_i$. By the end of the straightening phase, all particles will be located on these $k$ directrices.

We use an important basic subroutine, called *pulling procedure*. We assume to have a *chain* $Q$ of contracted particles, the first of which, $q$, is called the *pioneer*. Suppose $q$ intends to move to an unoccupied neighboring vertex $v$, "pulling" the whole chain with it. So, $q$ sends a message saying "Follow-Me" to the next particle $q'$ in the chain $Q$, and then it expands to $v$ and contracts again. When $q'$ receives the message, it forwards it to the next particle in $Q$, and moves to the position previously occupied by $q$, and so on. When the last particle of $Q$ has moved, it sends a message to its predecessor saying "Pulling-Complete", which is forwarded all the way to the pioneer, and the procedure ends.

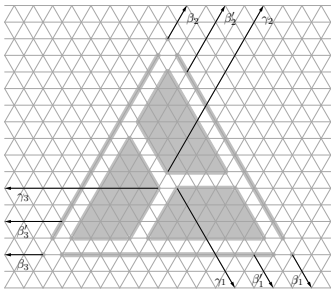
The idea of the straightening phase is that a pioneer $q_i$ will walk along each directrix $\gamma_i$, pulling particles onto it from the tree $T_i$ of the leader $p_i$ (executing the pulling procedure described above). While the pioneer is doing that, the leader remains in place, except perhaps for a few stages, when it is part of a chain of particles that is being pulled by the pioneer. Eventually, all the particles of $T_i$ will form a line segment on the directrix.

If $q_i$ encounters another particle $r$ on $\gamma_i$, belonging to some tree $T_j$, it "transfers" its role to $r$, and "claims" the subtree $T'_j$ of $T_j$ hanging from $r$, detaching $r$ from its parent. The next time the new pioneer $r$ has to pull a chain of particles, it will pull it from $T'_j$. For this reason, $r$ is called an *entry point* of the directrix. This algorithm is summarized in Figure 4.

If there is only $k = 1$ leader, there are no problems with the correctness of this algorithm. Suppose that $k > 1$: then, every time a pioneer advances along its directrix, it notifies its leader, who will synchronize with the other leaders (the details are in [15]). This is to ensure that the straightening of every tree proceeds at the same pace. As a consequence, the $k$ trees of the spanning forest, which initially are symmetric under a $k$-fold rotation of the plane, remain symmetric while the straightening proceeds.




**(a)** An unbreakably 2-symmetric shape with scale 5 with a minimal equivalent shape consisting of two adjacent faces and two edges



**(b)** An unbreakably 3-symmetric shape with scale 13 with a minimal equivalent shape consisting of a single face

**Figure 5** Subdivision into elements (gray blobs) of unbreakably $k$-symmetric shapes. The directrices, the backbone, and the co-backbone are also represented.

Because of this symmetry, no conflicts between different pioneers can ever arise. For instance, it is impossible for a leaf $f$ of a tree to be pulled along the chain led by a pioneer while another pioneer is trying to "transfer" to $f$. Also, the $k$ pulling procedures that are executed in the same step involve disjoint chains: indeed, the $k$ directrices are disjoint, and the subtrees hanging from different entry points are disjoint.

Let us prove that a system of $n$ particles executing this phase of the algorithm performs $O(n^2)$ moves in total. Observe that each pulling procedure causes a new particle to join a directrix, and so at most $n$ pulling procedures are performed. On the other hand, each pulling procedure involves at most $n$ particles, and causes each of them to perform a single expansion and a single contraction. The $O(n^2)$ bound follows, and the same bound on the number of rounds can be obtained similarly.

## 3.6 Role Assignment Phase

At the end of the straightening phase, the system forms $k$ equally long line segments, each of which contains a leader particle. If $k > 1$, it means that the shape $S_0$ that the particles originally formed was unbreakably $k$-symmetric. Due to Theorem 1, if this is the case, we have to assume that the "final shape" $S_F$ that the system has to form is also unbreakably $k$-symmetric.

In the role assignment phase, the particles determine the scale of the shape $S'_F$, equivalent to $S_F$, that they are actually going to form. Then, each particles is assigned an identifier describing which element of $S_F$ it is going to form in the shape composition phase.

Let $S'_F$ be a shape equivalent to $S_F$. The vertices of $G$ that lie in $S'_F$ are partitioned into *elements* as shown in Figure 5. Each vertex of $G$ that is in $S_F$ corresponds to a *super-vertex* of $S'_F$; the interior of each edge of $G$ contained in $S_F$ corresponds to a *super-edge* of $S'_F$; the interior of each face of $G$ contained in $S_F$ corresponds to a *super-triangle* of $S'_F$. The only exceptions are the central super-edge of an unbreakably 2-symmetric shape, which is further divided into two *partial super-edges*, and the central super-triangle of an unbreakably 3-symmetric shape, which is further divided into three (trapezoidal) *partial super-triangles*.

If there are $k > 1$ leaders, they have to partition the elements of $S'_F$ into $k$ equal parts, so that the $i$th leader will form one part, $(S'_F)_i$. Assume that the similarity transformation mapping $S_F$ to $S'_F$ is actually a homothety centered at the center of $S_0$. First the $i$th leader defines a *backbone ray* $\beta_i$ parallel to its directrix $\gamma_i$, as shown in Figure 5. If $k = 3$, it also

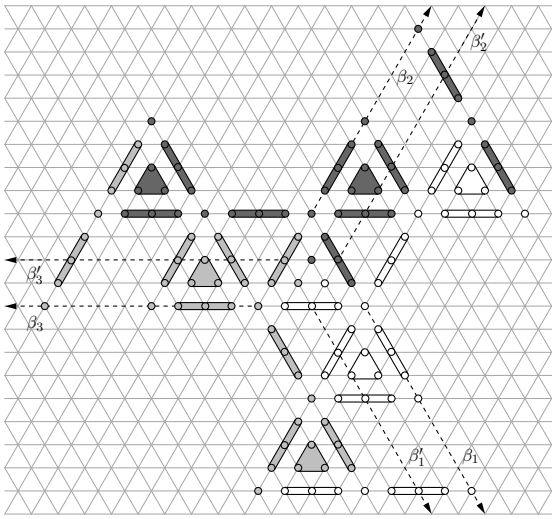



**Figure 6** The elements of an unbreakably 3-symmetric shape with a possible subdivision among leaders. Blobs of the same color represent elements selected by the same leader.

defines a co-backbone $\beta'_i$, as in Figure 5b. Each leader first selects for itself all the elements of $S'_F$ that intersect $\beta_i$ or $\beta'_i$ (the purpose of this selection will become clear in the next phase). Then, each leader repeatedly and deterministically selects an element of $S'_F$ that is adjacent to an element that it has already selected and that has not been selected by any leader, yet (see [15] for further details). As a result, $S'_F$ is partitioned into $k$ symmetric sub-shapes $(S'_F)_i$: one possible outcome is illustrated in Figure 6. Note that, since the above algorithm is deterministic, no explicit agreement between leaders is needed.

Let us focus on a single directrix and its leader. In the role assignment phase, the leader particle repeatedly changes its internal state and transfers the leadership to a neighbor (by sending it a message). So, the particles on the directrix can collectively compute any function that is computable by a Turing machine on a tape of length $n/k$: the leader is the head of the machine, and the particles are the cells of its tape. Hence, with standard techniques, the leader can do computations with binary numbers using the particles as bits. In particular, it can compute a scale $\lambda$ for $S'_F$ that is large enough for it to contain all $n$ particles, and small enough for it to be completely covered by them. Moreover, it can do so in $O(n^2)$ rounds (the details are in [15]). The number of vertices of $G$ covered by $S'_F$ may not be exactly $n$, but recall that each particle can occupy two vertices in its extended state: if $n$ is large enough compared to the size of $S_F$ (which is a constant $m$), a suitable $\lambda$ can indeed be found. Some particles will have to be extended in the final configuration, and these are marked with a special *Double* flag: these will be called *double particles*.

Then, the leader assigns a constant-size *identifier* to each element of $(S'_F)_i$ (there is a constant number of them), and assigns each particle the *role identifier* corresponding to the element it will contribute to forming in the next phase. To this end, the particles are subdivided into contiguous *chunks*, each of which is associated with an element of $(S'_F)_i$ and has the appropriate size (which can be computed "indirectly" by the leader, due to the Turing-machine analogy above). If $n$ is large enough, the leader can even place all the double particles in chunks corresponding to (partial) super-triangles (the special case in which $S_F$ consists only of edges is discussed in [15]).

## 3.7 Shape Composition Phase

In this phase, each of the $k$ leaders will guide its *team* of particles in the formation of all the elements of its sub-shape $(S'_F)_i \subseteq S'_F$. If $k > 1$, the leader preliminarily relocates its entire team from the directrix $\gamma_i$ to the backbone ray $\beta_i$. This can be done with the pulling procedure used in Section 3.5 and the Turing-machine technique of Section 3.6. Indeed, the leader knows the distance between $\gamma_i$ and $\beta_i$ in terms of $\lambda$, but it does not have enough internal memory to count to $\lambda$. So, it can use its team as a tape and count its steps in binary.

When the team is on $\beta_i$, it starts forming the elements of $(S'_F)_i$ one by one, coordinated by the leader. First the super-edges *adjacent* to the backbone ray $\beta_i$ are formed; then the super-vertices not on $\beta_i$ adjacent to those super-edges, then the super-edges not on $\beta_i$ adjacent to those super-vertices, etc. When all super-vertices and super-edges of $(S'_F)_i$ not on $\beta_i$ have been formed, the (partial) super-triangles are formed. Finally, the elements on $\beta_i$ are formed, from the closest to the center of $S'_F$ to the farthest.

The elements on $\beta_i$ are formed last because this ray is used by the leader to pull the entire "repository" of chunks wherever it has to go to form the next element $e$. Even though $(S'_F)_i$ may not be connected (as in Figure 6), the leader has enough particles in one chunk to count its steps while it pulls, and therefore to stop in the right position (the only exception is when only super-vertex chunks are left in the repository: this case is discussed in [15]).

Once the repository is in position to reach $e$, the leader moves the corresponding chunk $C_e$ from the repository along a path $W$ of already formed super-edges and super-vertices of $(S'_F)_i$ (by simply swapping states of particles). This is easy to do, since there are no double particles in $W$: they are all in the (partial) super-triangles. When $C_e$ becomes adjacent to $e$, the leader pulls it into position, also pulling $W$ (which, as a result, goes back in its place) and the rest of the repository. Then the leader leaves $e$ and goes back to $\beta_i$ through $W$.

Note that, when $e$ is a (partial) super-triangle, the leader may have to use the Turing-machine technique to count its steps while forming it, so to know when it has reached a corner of $e$. If $e$ contains double particles, the leader first pulls all of $C_e$ into $e$, and then it leaves $e$. When the leader is gone, the first particle of $C_e$ keeps pulling part of the chunk to let the double particles expand and finally cover all of $e$.

To prove the correctness of this algorithm, it is crucial to observe that it is impossible for two different teams to come into contact and interfere with each other's movements: this is because they are confined to move within different regions of $G$ throughout the phase. Note that selecting all the elements that intersect its backbone ray and its co-backbone ray (see Section 3.6) gives a leader and its team free range to move between their directrix and their backbone ray during the preliminary relocation step. This is true also if one team has already started forming its elements while another is still moving from the directrix to the backbone.

Let us count the total number of moves performed in this phase. When a leader relocates its team onto the backbone, it pulls all the particles at most $O(n)$ times, and the total number of moves is at most $O(n^2)$. Then, in order to form one element of $S'_F$, a leader may have to pull at most $O(n)$ particles (i.e., the repository) for at most $O(n)$ times along the backbone to get the chunk into position: this yields at most $O(n^2)$ moves. Then it has to pull at most $O(n)$ particles for a number of times that is equal to the size of the element of $S'_F$, which is $O(n)$. Since the number of elements of $S'_F$ is bounded by a constant, this amounts to at most $O(n^2)$ moves, again. All other operations involve only message exchanges and no movements. The $O(n^2)$ upper bound follows, and the same bound on the number of rounds is obtained similarly. (The proof of the $\Theta(m^3)$ bound on $n$ in the theorem below is in [15].)

▶ **Theorem 3.** *Let $P$ be a system of $n$ particles forming a simply connected shape $S_0$ at stage 0. Let $S_F$ be a shape of constant size $m$ that is unbreakably $k$-symmetric if $S_0$ is unbreakably $k$-symmetric. If all particles of $P$ execute the universal shape formation algorithm with input*

*a representation of the final shape $S_F$, and if $n$ is at least $\Theta(m^3)$, then there is a stage, reached after $O(n^2)$ rounds, where $P$ forms a shape equivalent to $S_F$. The total number of moves performed by $P$ up to this stage is $O(n^2)$, which is asymptotically worst-case optimal; particles no longer move afterwards.*

## 4 Generalization to All Computable Infinite-Resolution Shapes

In the previous section we have shown that, given a shape $S_F$ of constant size $m$, a system of $n$ particles can form a shape geometrically similar to $S_F$ (i.e., essentially a scaled-up copy of $S_F$) starting from any simply connected configuration $S_0$, provided that $S_F$ is unbreakably $k$-symmetric if $S_0$ is, and provided that $n$ is large enough compared to $m$. We only determined a bound of $\Theta(m^3)$ for the minimum $n$ that guarantees the formability of $S_F$. We could improve it to $\Theta(m)$ by letting the Double particles be in any chunk and adopting a slightly more sophisticated pulling procedure in the last phase. We may wonder if this modification would make our bound optimal.

When discussing the role assignment phase, when the particles are arranged along straight lines, we have argued that the system can compute any predicate that is computable by a Turing machine on a tape of limited length. If we allow the particles to move back and forth along these lines to simulate registers, we only need a (small) constant number of particles to implement a full-fledged Turing machine with an infinite tape. So, in the role assignment phase, we are actually able to compute any Turing-computable predicate (although we would have to give up our upper bounds of $O(n^2)$ moves and rounds).

With this technique, we are not only able to replace our $\Theta(m^3)$ with the best possible asymptotic bound in terms of $m$, but we have a universal shape formation algorithm that, for every $n$ and every $S_F$, lets the system determine if $n$ particles are enough to form a shape geometrically similar to $S_F$. This is done by examining all the possible connected configurations of $n$ particles and searching for one that matches $S_F$, which is of course a Turing-computable task.

Taking this idea even further, we can extend our notion of shape to its most general form. Recall that the shapes considered in [8] were sets of "full" triangles: when a shape is scaled up, all its triangles are scaled up and become larger full triangles. In this paper, we extended the notion of shape to sets of full triangles and edges: when an edge is scaled up, it remains a row of points. Of course, we can think of shapes that are not modeled by full triangles or edges, but behave like fractals when scaled up. For instance, we may want to include discretized copies of the Sierpinski triangle as "building blocks" of our shapes, alongside full triangles and edges. Scaling up these shapes causes their "resolution" to increase and makes finer details appear inside them. Clearly, the set of all the scaled-up and discretized copies of a shape made up of full triangles, edges, and Sierpinski triangles is Turing-computable.

Generalizing, we can replace our usual notion of geometric similarity between shapes with any Turing-computable equivalence relation $\sim$. Then, the shape formation problem with input a shape $S_F$ asks to form any shape $S'_F$ such that $S_F \sim S'_F$. This definition of shape formation problem includes and greatly generalizes the one studied in this paper, and even applies to scenarios that are not of a geometric nature. Nonetheless, this generalized problem is still solvable by particles, thanks to the technique outlined above.

## References

1 H. Ando, I. Suzuki, and M. Yamashita. Formation and agreement problems for synchronous mobile robots with limited visibility. In *Proc. of ISIC*, pages 453–460, 1995.

2 S. Cannon, J.J. Daymude, D. Randall, and A.W. Richa. A Markov chain algorithm for compression in self-organizing particle systems. In *Proc. of PODC*, pages 279–288, 2016.

3 G. Chirikjian. Kinematics of a metamorphic robotic system. In *Proc. of ICRA*, pages 1:449–1:455, 1994.

4 S. Das, P. Flocchini, N. Santoro, and M. Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. *Distrib. Comput.*, 28(2):131–145, 2015.

5 J.J. Daymude, Z. Derakhshandeh, R. Gmyr, A. Porter, A.W. Richa, C. Scheideler, and T. Strothmann. On the runtime of universal coating for programmable matter. In *Proc. of DNA*, pages 148–164, 2016.

6 J.J. Daymude, R. Gmyr, A.W. Richa, C. Scheideler, and T. Strothmann. Improved leader election for self-organizing programmable matter. *arXiv*, 2017. URL: `https://arxiv.org/abs/1701.03616`.

7 Z. Derakhshandeh, R. Gmyr, A.W. Richa, C. Scheideler, and T. Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proc. of NanoCom*, pages 21:1–21:2, 2015.

8 Z. Derakhshandeh, R. Gmyr, A.W. Richa, C. Scheideler, and T. Strothmann. Universal shape formation for programmable matter. In *Proc. of SPAA*, pages 289–299, 2016.

9 Z. Derakhshandeh, R. Gmyr, A.W. Richa, C. Scheideler, and T. Strothmann. Universal coating for programmable matter. *Theor. Comput. Sci.*, 671:56–68, 2017.

10 Z. Derakhshandeh, R. Gmyr, T. Strothmann, R.A. Bazzi, A.W. Richa, and C. Scheideler. Leader election and shape formation with self-organizing programmable matter. In *Proc. of DNA*, pages 117–132, 2015.

11 S. Dolev, S. Frenkel, M. Rosenbli, P. Narayanan, and K.M. Venkateswarlu. In-vivo energy harvesting nano robots. In *Proc. of ICSEE*, pages 1–5, 2016.

12 P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theor. Comput. Sci.*, 407(1):412–447, 2008.

13 N. Fujinaga, Y. Yamauchi, H. Ono, S. Kijima, and M. Yamashita. Pattern formation by oblivious asynchronous mobile robots. *SIAM J. Comput.*, 44(3):740–785, 2016.

14 G.A. Di Luna, P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta. Line recovery by programmable particles. In *Proc. of ICDCN*, to appear.

15 G.A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. Shape formation by programmable particles. *arXiv*, 2017. URL: `https://arxiv.org/abs/1705.03538`.

16 O. Michail, G. Skretas, and P.G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. In *Proc. of ICALP*, pages 136:1–136:15, 2017.

17 A. Naz, B. Piranda, J. Bourgeois, and S.C. Goldstein. A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots. In *Proc. of NCA*, pages 254–263, 2016.

18 M.J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Nat. Comput.*, 13(2):195–224, 2014.

19 P.W. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

20 M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

21 N. Schiefer and E. Winfree. Universal computation and optimal construction in the chemical reaction network-controlled tile assembly model. In *Proc. of DNA*, pages 34–54, 2015.

22 I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999.

23 T. Toffoli and N. Margolus. Programmable matter: concepts and realization. *Physica D*, 47(1):263–272, 1991.

24 J.E. Walter, J.L. Welch, and N.M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distrib. Comput.*, 17(2):171–189, 2004.