Leader Election and Shape Formation with Self-Organizing Programmable Matter

Zahra Derakhshandeh[1†], Robert Gmyr[2‡], Thim Strothmann[2‡],
Rida Bazzi[1], Andréa W. Richa[1†], Christian Scheideler[2‡]

[1] Computer Science, CIDSE
Arizona State University, USA,
{zderakhs,bazzi,aricha}@asu.edu
[2] Department of Computer Science,
University of Paderborn, Germany,
{gmyr,thim}@mail.upb.de, scheideler@upb.de

Abstract. In this paper we consider programmable matter consisting of simple computational elements, called *particles*, that can establish and release bonds and can actively move in a self-organized way, and we investigate the feasibility of solving fundamental problems relevant for programmable matter. As a model for such self-organizing particle systems, we will use a generalization of the geometric amoebot model first proposed in [21]. Based on the geometric model, we present efficient local-control algorithms for leader election and line formation requiring only particles with constant size memory, and we also discuss the limitations of solving these problems within the general amoebot model.

1 Introduction

A central problem for programmable matter is shape formation, and various solutions have already been found for that problem using different approaches like DNA tiles [34], moteins [14], or nubots [39]. We are studying shape formation using the amoebot model which was first proposed in [21]. In order to determine how decentralized shape formation can be handled, we are particularly interested in the connection between leader election and shape formation. In the leader election problem we are given a set of particles, and the problem is to select one of these as the leader. Many problems like the consensus problem (all particles have to agree on some output value) can easily be solved once the leader election problem can be solved. The same has also been observed for shape formation, as most shape formation algorithms depend on some seed element. However, the question is whether shape formation can even be solved in circumstances where leader election is not possible. The aim of this paper is to shed some light on the dependency between leader election and shape formation by focusing on the special problem of forming a line of particles. Before we present our results, we first give a formal definition of the model and the problems we intend to study.

† Supported in part by the NSF under Awards CCF-1353089 and CCF-1422603.
‡ Supported in part by DFG grant SCHE 1592/3-1.

1.1 Models

We use two models throughout this work. Firstly, we consider a generalization of the amoebot model [21] which abstracts from any geometry information. We call this model the *general amoebot model*. Secondly, we consider a model that is essentially equivalent to the original amoebot model presented in [21] but is defined based on the general amoebot model. We refer to this second model as the *geometric amoebot model*.

In the *general amoebot model*, programmable matter consists of a uniform set of simple computational units called particles that can move and bond to other particles and use these bonds to exchange information. The particles act asynchronously and they achieve locomotion by expanding and contracting, which resembles the behavior of amoeba.

As a base of this model, we assume that we have a set of particles that aim at maintaining a connected structure at all times. This is needed to prevent the particles from drifting apart in an uncontrolled manner like in fluids and because in our case particles communicate only via bonds. The shape and positions of the bonds of the particles mandate that they can only assume discrete positions in the particle structure. This justifies the use of a possibly infinite, undirected graph $G = (V, E)$, where V represents all possible positions of a particle (relative to the other particles in their structure) and E represents all possible transitions between positions.

Each particle occupies either a single node or a pair of adjacent nodes in G, i.e., it can be in two different *shapes*, and every node can be occupied by at most one particle. Two particles occupying adjacent nodes are *connected*, and we refer to such particles as *neighbors*. Particles are *anonymous* but the bonds of each particle have unique labels, which implies that a particle can uniquely identify each of its outgoing edges. Each particle has a local memory, and any pair of connected particles has a shared memory that can be read and written by both particles.

Particles move through *expansions* and *contractions*: If a particle occupies one node (i.e., it is *contracted*), it can expand to an unoccupied adjacent node to occupy two nodes. If a particle occupies two nodes (i.e., it is *expanded*), it can contract to one of these nodes to occupy only a single node. Performing movements via expansions and contractions has various advantages. For example, it would easily allow a particle to abort a movement if its movement is in conflict with other movements. A particle always knows whether it is contracted or expanded and this information will be available to neighboring particles. In a *handover*, two scenarios are possible: a) a contracted particle p can "push" a neighboring expanded particle q and expand into the neighboring node previously occupied by q, forcing q to contract, or b) an expanded particle p can "pull" a neighboring contracted particle q to a cell occupied by it thereby expanding that particle to that cell, which allows p to contract to its other cell. The ability to use a handover allows the system to stay connected while particles move (e.g., for particles moving in a worm-like fashion). Note that while expansions and contractions may represent the way particles physically move in space,

they can also be interpreted as a particle "looking ahead" and establishing new logical connections (by expanding) before it fully moves to a new position and severs the old connections it had (by contracting).

Summing up over all assumptions above, the *state* of a particle is uniquely determined by its shape, the contents of its local memory, the edges it has to neighboring particles, the contents of their shared memory (which may allow a particle to obtain further information about the neighboring particles beyond their shape), and finally the shape of the neighboring particles. The *state of the particle system* (or short, *system state*) is defined as the combination of all particle states. We say a particle system in a system state in which the particle occupy a set of nodes $A \subseteq V$ is *connected* if the graph $G|_A$ induced by A is connected. We assume the standard asynchronous computation model, i.e., only one particle can be active at a time. Whenever a particle is active, it can perform an *action* (governed by some fixed, finite size program controlling it) consisting of a finite amount of computation (involving its local memory, the shared memories with its neighboring particles, and random bits) followed by no or a single movement. Hence, a *computation* of a particle system is a potentially infinite sequence of actions $A_1, A_2, \ldots$ based on some initial system state s_0, where action A_i transforms system state s_{i-1} into system state s_i. The (parallel) time complexity of a computation is usually measured in *rounds*, where a round is over once every particle has been given the chance to perform at least one action.

Let $\mathcal{S}$ be the set of all system states in which the particle system is connected. In general, a *computational problem* P for the particle system is specified by a set $\mathcal{S}' \subseteq \mathcal{S}$ of permitted initial system states and a mapping $F : \mathcal{S}' \to 2^{\mathcal{S}}$, where $F(s) \subseteq \mathcal{S}$ determines the set of permitted *final* states for any initial state $s \in \mathcal{S}'$. A particle system *solves* problem $P = (\mathcal{S}', F)$ if for any initial system state $s \in \mathcal{S}'$, all computations of the particle system eventually reach a system state in $F(s)$ without losing connectivity, and whenever such a system state is reached for the first time, the system stays in $F(s)$. If for all computation a final state is reached in which all particles decided to halt (i.e., they decided not to perform any further actions, irrespective of future events), then the particle system is also said to *decide* problem P. Note that being in a final state does not necessarily mean that all particles decided to halt. If $\mathcal{S}' = \mathcal{S}$, so *any* initial state is permitted (including arbitrary faulty states, as long as the particle system is connected), then a particle system solving P is also said to be *self-stabilizing*. It is well-known that in general a distributed system solving a problem P cannot decide it and also be self-stabilizing because if so, it would often be possible to come up with an initial state s where a member of the system decides to halt prematurely, disallowing the system to eventually reach a state in $F(s)$.

Besides the general amoebot model, we will also consider the *geometric amoebot model*. The geometric amoebot model is a specific variant of the general amoebot model in which the underlying graph G is defined to be the equilateral triangular graph G_{eqt} (see Figure 1), and the bonds of the particles are labeled in a consecutive way in clockwise orientation around a particle so that every

particle has the same sense of clockwise orientation. However, we do not assume that the labeling is uniform, so the particles do not necessarily share a common sense of direction in the grid.

Fig. 1. The left part shows an example of a particle structure in the geometric amoebot model. A contracted particle is depicted as a black dot, and an expanded particle is depicted as two black dots connected by an edge. The right part shows a particle structure with 3 borders. The outer border is shown as a solid line and the two inner borders are shown as dashed lines.

1.2 Problems

In this paper we consider the following two problems. For both problems we define the set of initial system states as the set of all states such that the particle system is connected and all memories are empty.

For the *leader election problem* the set of final system states contains any state in which the particles form a connected structure and exactly one particle is a leader (i.e., only this particle is in a leader state while the remaining particles are in a non-leader state). Our goal will be to come up with a distributed algorithm that allows a particle system to decide the leader election problem. Note that the leader election problem is well defined for both the general amoebot model and the geometric amoebot model.

In a *shape formation problem*, the set of final states consists of those system states where the particle structure forms the desired shape. As a specific example of a shape formation problem, we consider the *line formation problem*. In the geometric amoebot model, the shape the particles have to form is a straight line in the equilateral triangular grid and all particles have to be contracted in a final system state. Of course, in the general amoebot model a straight line is not well-defined. Hence, for this model the set of final states for the line formation problem is defined to consist of all system states in which the particles form a simple path in G.

Throughout the paper, we assume for the sake of simplicity that in an initial state all particles are contracted. Our algorithms can easily be extended to dispose of this assumption.

1.3 Our Contributions

In this paper we focus on the problem of solving leader election and shape formation for particles with *constant memory*. For shape formation, we just focus on the already mentioned line formation problem.

For the general amoebot model, we can show that neither leader election nor shape formation can be decided by any distributed algorithm. Suppose that there is a distributed algorithm solving the line formation problem in the general amoebot model (when starting in a well-initialized state). Since in this case it is possible to decide when $G|_{A'}$ forms a line, it is also possible to design a protocol that solves the leader election problem: once the line has been formed, its two endpoints contend for leadership using tokens with random bits sent back and forth until one of them wins. On the other hand, one can deduce from [28] that in the general amoebot model there is no distributed algorithm that can decide when a leader has been elected (with any reasonable success probability). More concretely, in [28] the authors show that for the ring of anonymous nodes there is no algorithm that can correctly decide the leader election problem (or in their words, that can solve the leader election problem with distributive termination) with any probability $\alpha > 0$, i.e., for any algorithm in which the particles are guaranteed to halt, the error probability is unbounded. Since in the general amoebot model G can be any graph, we can set G to be a ring whose size is the number of particles and the result of [28] is directly applicable. Hence, there cannot be a distributed algorithm deciding the line formation problem (with any reasonable success probability) in the general amoebot model, and therefore not even an algorithm for solving it since a protocol solving the problem could easily be transformed into a protocol deciding it. However, for the *geometric* amoebot model we show that there is a distributed algorithm that can decide the leader election problem, i.e., at the end we have exactly one leader and the leader knows that it is the only leader left. Moreover, the runtime for our leader election algorithm is worst-case optimal in a sense that it needs at most $O(L)$ rounds on expectation, where L is the maximum length of a border between the particle structure and an empty region (inside or outside of it) in G_{eqt}. Based on the leader election algorithm, we present a distributed algorithm that solves the line formation problem. Both algorithms assume that the system is in a well-initialized state. It would certainly be desirable to have algorithms that can tolerate any initial state, but at the end of the paper we show that there are certain limitations to solving leader election and line formation in a self-stabilizing fashion.

1.4 Related Work

Many approaches related to programmable matter have recently been proposed. One can distinguish between active and passive systems. In passive systems the particles either do not have any intelligence at all (but just move and bond based on their structural properties or due to chemical interactions with the environment), or they have limited computational capabilities but cannot control

their movements. Examples of research on *passive systems* are DNA computing [1, 8, 14, 20, 37], tile self-assembly systems in general (e.g., see the surveys in [22, 34, 38]), population protocols [3], and slime molds [9, 32]. We will not describe these models in detail as they are only of little relevance for our approach. On the other hand in *active systems*, computational particles can control the way they act and move in order to solve a specific task. Robotic swarms, and modular robotic systems are some examples of active programmable matter systems.

In the area of *swarm robotics* it is usually assumed that there is a collection of autonomous robots that have limited sensing, often including vision, and communication ranges, and that can freely move in a given area. They follow a variety of goals, for example graph exploration (e.g., [23]), gathering problems (e.g., [2, 16]), shape formation problems (e.g., [24, 35]), and to understand the global effects of local behavior in natural swarms like social insects, birds, or fish (see e.g., [7, 11]). Surveys of recent results in swarm robotics can be found in [30, 33]; other samples of representative work can be found in e.g., [4, 6, 17–19, 27, 31]. While the analytical techniques developed in the area of swarm robotics and natural swarms are of some relevance for this work, the individual units in those systems have more powerful communication and processing capabilities than in the systems we consider.

The field of *modular self-reconfigurable robotic systems* focuses on intra-robotic aspects such as the design, fabrication, motion planning, and control of autonomous kinematic machines with variable morphology (see e.g., [25, 40]). *Metamorphic robots* form a subclass of self-reconfigurable robots that share some of the characteristics of our geometric model [15]. The hardware development in the field of self-reconfigurable robotics has been complemented by a number of algorithmic advances (e.g., [10, 35, 36]), but so far mechanisms that automatically scale from a few to hundreds or thousands of individual units are still under investigation, and no rigorous theoretical foundation is available yet.

The *nubot* model [12, 13, 39] by Woods et al. aims at providing the theoretical framework that would allow for a more rigorous algorithmic study of biomolecular-inspired systems, more specifically of self-assembly systems with active molecular components. While bio-molecular inspired systems share many similarities with our self-organizing particle systems, there are many differences that do not allow us to translate the algorithms and other results under the nubot model to our systems — e.g., there is always an arbitrarily large supply of "extra" particles that can be added to the system as needed, and the system allows for an additional (non-local) notion of rigid-body movement.

2 Leader Election in the Geometric Amoebot Model

In this section we show how the leader election problem can be decided in the geometric amoebot model. Our approach organizes the particle system into a set of cycles and executes an algorithm on each cycle independently. For simplicity and ease of presentation we first assume that particles have a global view of the cycle they are part of, that agents act synchronously, and that their local memory is

unbounded. However, in the local-control protocol none of these assumptions are needed. In particular, the particles only require a constant amount of memory. In Section 2.5 we highlight some of the techniques used in the local-control protocol, which relies heavily on token passing. However, due to space constraints the full local-control protocol cannot be presented in detail.

2.1 Organization into Cycles

Let $A \subseteq V$ be any initial distribution of contracted particles such that $G_{\text{eqt}}|_A$ is connected. Consider the graph $G_{\text{eqt}}|_{V \setminus A}$ induced by the unoccupied nodes in G_{eqt}. We call a connected component of $G_{\text{eqt}}|_{V \setminus A}$ an *empty region*. Let $N(R)$ be the neighborhood of an empty region R in G_{eqt}. Then all nodes in $N(R)$ are occupied and we call the graph $G_{\text{eqt}}|_{N(R)}$ a *border*. Since $G_{\text{eqt}}|_A$ is a connected finite graph, exactly one empty region has infinite size while the remaining empty regions have finite size. We define the border corresponding to the infinite empty region to be the unique *outer border* and refer to a border that corresponds to a finite empty region as an *inner border*, see Figure 1.

The particles occupying a border can instantly (i.e., without communication) organize themselves into a cycle using only local information: Consider a border corresponding to an empty region R. Let p be a particle occupying a node v of the border. By definition there exists a non-occupied node $w \in R$ that is a adjacent to v in the graph G_{eqt}. The particle p iterates over the neighboring nodes of v in clockwise orientation around v starting at w. Consider the first occupied node it encounters; the particle occupying that node is the successor of p in the cycle corresponding to that border. Analogously, p finds its predecessor in the cycle by traversing the neighborhood of v in counter-clockwise orientation.

Note, that a particle can belong to up to three borders at once. Furthermore, a particle cannot locally decide whether two empty regions it sees (i.e., maximal connected components of non-occupied nodes in the neighborhood of v) are distinct. We circumvent these problems by letting a particle treat each empty region in its local view as distinct. For each such empty region, a particle executes an independent instance of the same algorithm. Hence, we say a particle acts as a number of distinct *agents*. For each of its agents a particle determines the predecessor and successor as described above. This effectively connects the set of all agents into disjoint cycles as depicted in Figure 2. Observe that from a global perspective the cycle of the outer border is oriented clockwise while a cycle of an inner border is oriented counter-clockwise. This is a direct consequence of the way the predecessors and successors of an agent are defined.

2.2 Algorithm

The leader election algorithm operates independently on each cycle. At any given time, some subset of agents on a cycle will consider themselves *candidates*, i.e. potential leaders of the system. Initially, every agent considers itself a candidate. Between any two candidates on a cycle there is a (possibly empty) sequence of non-candidate agents. We call such a sequence a *segment*. For a candidate c

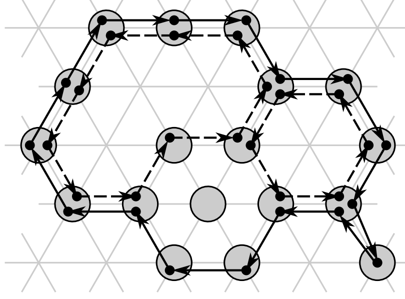

Fig. 2. The depicted particle system is the same as in the right part of Figure 1. In this figure particles are depicted as gray circles. The black dots inside of a particle represent its agents. As in Figure 1 the outer border is solid and the two inner borders are dashed.

we refer to the segment coming after c in the direction of the cycle as $seg(c)$ and refer to its length by $|seg(c)|$. We refer to the candidate coming after c as the *succeeding candidate* ($succ(c)$) and to the candidate coming before c as the *preceding candidate* ($pred(c)$) (see Figure 3). We drop the c in parentheses if it is clear from the context. We define the distance $d(c_1, c_2)$ between candidates c_1 and c_2 as the number of agents between c_1 and c_2 when going from c_1 to c_2 in *direction* of the cycle. We say a candidate c_1 *covers* a candidate c_2 (or c_2 *is covered by* c_1) if $|seg(c_1)| > d(c_2, c_1)$ (see Figure 3). The leader election

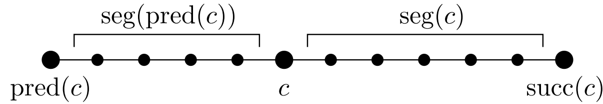

Fig. 3. A cutout from a cycle that is oriented to the right. Non-candidate agents are small black dots, candidates are bigger dots. The candidate c covers $pred(c)$ since $|seg(c)| > d(pred(c), c)$.

progresses in *phases*. In each phase, each candidate executes Algorithm 1. A phase consists of three synchronized *subphases*, i.e., agents can only progress to the next subphase once all agents have finished the current subphase.

Consider the execution of Algorithm 1 by a candidate c. If the algorithm returns "not leader" then c revokes its candidacy and becomes part of a segment. If the algorithm returns "leader", c will become the leader of the particle system. The transferal of candidacy in subphase 2 means that c withdraws its own candidacy but at the same time promotes the agent at position *pos* (i.e., $succ(c)$ in subphase 1) to be a candidate. Once a candidate becomes a leader it broadcasts this information such that all particles can halt.

Algorithm 1 Leader Election for a candidate c

```
Subphase 1:
pos ← position of succ(c)
if covered by any candidate or |seg(c)| < |seg(pred(c))| then
    return not leader

Subphase 2:
if coin flip results in heads then
    transfer candidacy to agent at pos

Subphase 3:
if only candidate on border then
    if outside border then
        return leader
    else
        return not leader
```

2.3 Correctness

In order to show the correctness of our algorithm, we show that it satisfies the following conditions, that relate to the entire particle system (not just a single cycle):

1. *Safety*: There always exists at least one candidate.
2. *Liveness*: In each phase if there is more than one candidate, at least one candidate withdraws leadership with a probability that is bounded below by a positive constant.

Lemma 1. *Algorithm 1 satisfies the safety condition.*

Proof. We will show by induction that on the cycle associated with the outer border there will always be at least one candidate. Initially, this holds trivially. So assume that it holds before a phase. Let c be the candidate with the longest segment. Then there is no candidate covering c and also $|seg(c)| < |seg(pred(c))|$ cannot be true. Hence, c will not withdraw candidacy in subphase 1. In subphase 2, the candidacy of c might be transferred but will not vanish. Let c' be the agent that received the candidacy if it was transferred and $c' = c$ otherwise. In subphase 3, c' will not withdraw candidacy because it lies on the outer border. Hence, there is still a candidate after the phase. □

Lemma 2. *Algorithm 1 satisfies the liveness condition.*

Proof. Assume that there are two or more candidates in the system. First we consider the case that there is a cycle with two or more candidates. If there are segments of different lengths on that cycle, we have $|seg| < |seg(pred)|$ for at least one candidate which will therefore withdraw its candidacy in subphase 1. If all segments are of equal length, we have that in subphase 2 with probability at least $\frac{1}{4}$ there is a candidate c that transfers candidacy while $succ(c)$ does

not. Hence, the number of candidates is reduced with probability at least $\frac{1}{4}$. Now consider the case that all cycles have at most one candidate. Then there is a cycle corresponding to an inner border that has exactly one candidate. That candidate will withdraw candidacy in subphase 3 and thereby reduce the number of candidates in the system. □

The following Theorem is a direct consequence of Lemmas 1 and 2.

Theorem 1. *Algorithm 1 decides the leader election problem in the geometric amoebot model.*

2.4 Runtime Analysis

For a cycle of agents let L be the length of the cycle and let l_i be the longest segment length before phase i of the execution of Algorithm 1. We define $l_i = L$ if there is no candidate on the cycle. It is easy to see that if $l_i \geq L/2$ then in phase $i+1$ either the leader is elected (outer border) or all candidates on the cycle vanish (inner border). For the case $l_i < L/2$, Lemma 3 provides the key insight of our analysis.

Lemma 3. *For any phase i such that $l_i < L/2$ it holds $l_{i+1} \geq l_i$ in any case and $l_{i+1} \geq 2l_i$ with probability at least $1/4$.*

Let $L_{\max}$ be the length of the longest cycle in the particle system. Based on Lemma 3 it is easy to see that under complete synchronization of subphases and with the agents having a global view of the cycle, our algorithm requires on expectation $O(\log(L_{\max}))$ phases to elect a leader. For now assume that our algorithm can be realized as a local-control protocol such that phase i requires $O(l_i)$ rounds. Theorem 2 gives a bound on the number of rounds required by the algorithm based on this assumption. The theorem also also holds for the local-control protocol given the definition of a round from Section 1.1.

Theorem 2. *Algorithm 1 requires $O(L_{max})$ rounds on expectation.*

Proof. Let the random variable X_i describe the number of rounds during the execution of Algorithm 1 such that $l_i \in [2^{i-1}, 2^i)$. Then, under the assumption that phase i requires $O(l_i)$ rounds, the total runtime of our algorithm is

$$T = \sum_{i=1}^{\lceil \log(L_{\max}) \rceil} X_i \cdot O(2^i).$$

Since $E(X_i) \leq 4$ due to Lemma 3, the expected runtime is

$$E(T) \leq \sum_{i=1}^{\lceil \log(L_{\max}) \rceil} E(X_i) \cdot O(2^i) = O(L_{\max}).$$

□

Note that subphase 1 of the algorithm is not important in terms of correctness. However, it is crucial to achieve a linear runtime in expectation. If agents would only execute subphases 2 and 3, the runtime would degrade to $O(L_{\max} \log L_{\max})$.

2.5 Asynchronous Local-Control Protocol

Here we present some details on how specific parts of the algorithm can be realized as an asynchronous local-control protocol. We focus on the realization of solitude verification and the inner outer border test of subphase 3.

Solitude Verification A candidate that wants to determine whether it is the only candidate left, tests if its segment ends in another candidate or in itself. To do so, it enforces its own orientation on all agents in its segment. Thereby, every agent in the segment is able to determine the direction of its outgoing edge in direction of the cycle. These edge directions can be seen as vectors in the two dimensional plane and in case the segment is the whole cycle, the vectors cancel out component wise (see Figure 4). By a simple token passing scheme agents will try match their edge directions component wise with an edge direction in the opposing direction from another agent. Finally, the candidate inspects the segment and if all agents are matched it is the only candidate left on the cycle.

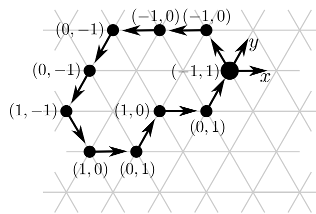

Fig. 4. An example of solitude verification: the candidate (shown slighly bigger) has enforced its orientation (x and y arrows) on all agents. All non-candidate agents determined the offset to the succeeding agent in direction of the cycle (arrows and numbers at nodes).

Inner Outer Border Test The last candidate of a cycle can decide whether its cycle corresponds to an inner or the outer border as follows. A cycle corresponding to an inner border has counter-clockwise rotation while a cycle corresponding to the outer border has clockwise rotation, see Figure 2. The candidate sends a token along the cycle that sums the angles of the turns the cycle takes, see Figure 5. When the token returns to the candidate its value represents the external

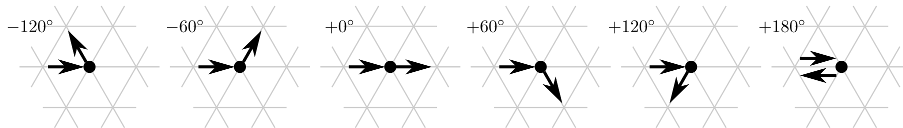

Fig. 5. The angle between the directions a token enters and exits an agent.

angle of the polygon corresponding to the cycle while respecting the rotation of the cycle. So it is $-360°$ for an inner border and $360°$ for the outer border. The token can represent the angle as an integer k such that the angle is $k \cdot 60°$. Furthermore, to distinguish the two possible final values of k it is sufficient to store the k modulo 5, so that the token only requires 3 bits of memory.

3 Line Formation in the Geometric Amoebot Model

Next we consider the line formation problem in the geometric amoebot model. We assume that initially we have an arbitrary connected structure of contracted particles with a unique leader. The leader is used as the starting point for the line of particles and specifies the direction in which this line will grow. As the line grows, every particle touched by the line that is already in a valid line position becomes part of the line. Any other particle connected to the line becomes the root of a tree of particles. Every root aims at traveling around the line in a clockwise manner until it joins the line. As a root particle moves, the other particles in its tree follow in a worm-like fashion (i.e., via a series of handover operations)[3].

Before we give a detailed description of the algorithm, we provide some preliminaries. We distinguish for the state of a particle between *inactive*, *follower*, *root*, and *retired* (or halted). Initially, all particles are *inactive*, except the leader particle, which is always in a *retired* state. In addition to its state, each particle p may maintain a constant number of *flags* in its shared memory. For an expanded particle, we denote the node the particle last expanded into as the *head* of the particle and call the other occupied node its *tail*: In our algorithm, we assume that every time a particle contracts, it contracts out of its tail.

The spanning forest algorithm, given in Algorithm 2, is a basic building block we use for shape formation problems. This algorithm aims at organizing the particles as a spanning forest, where the particles that represent the roots of the trees determine the direction of movement, whom the remaining particles follow. Each particle p continuously runs Algorithm 2 until p becomes retired. If particle p is a follower, it stores a flag $p.parent$ in its shared memory corresponding to the edge adjacent to its parent p' in the spanning forest (any particle q can then easily check if p is a child of q). If p is the leader particle, then it sets the flag $p.linedir$ in the shared memories corresponding to two of its edges in opposite directions (i.e., an edge i and the edge that appears three positions after i in clockwise order), denoting that it would like to extend the line through the directions given by these edges.

We have the following theorem, where *work* is defined as the number of expansions and contractions executed by all particles in the system:

Theorem 3. *Algorithm 2 solves the line formation problem in worst-case optimal $O(n)$ number of rounds and $O(n^2)$ work.*

[3] For a simulation video of the Line Formation Algorithm please see http://sops.cs.upb.de .

Algorithm 2 Line Formation Algorithm

SpanningForest (p):
Particle p acts as follows, depending on its current state:

inactive: If p is connected to a retired particle, then p becomes a *root* particle. Otherwise, if an adjacent particle p' is a root or a follower, p sets the flag $p.parent$ on the shared memory corresponding to the edge to p' and becomes a *follower*. If none of the above applies, it remains inactive.

follower: If p is contracted and connected to a retired particle, then p becomes a *root* particle. Otherwise, it considers the following three cases: (i) if p is contracted and p's parent p' (given by the flag $p.parent$) is expanded, then p expands in a handover with p', adjusting $p.parent$ to still point to p' after the handover; (ii) if p is expanded and has a contracted child particle p', then p executes a handover with p'; (iii) if p is expanded, has no children, and p has no inactive neighbor, then p contracts.

root: Particle p may become *retired* following CheckRetire (p). Otherwise, it considers the following three cases: (i) if p is contracted, it tries to expand in the direction given by LineDir(p); (ii) If p is expanded and has a child p', then p executes a handover contraction with p'; (iii) if p is expanded and has no children, and no inactive neighbor, then p contracts.

retired: p performs no further action.

CheckRetire (p):
  if p is a contracted root **then**
    if p has an adjacent edge i to p' with a flag $p'.linedir$, where p' is retired **then**
      Let i' be the edge opposite to i in clockwise order
      p sets the flag $p.linedir$ in the shared memory of edges i and i'
      p becomes *retired*.

LineDir(p):
  Let i be the label of an edge connected to a retired particle.
  while edge i points to a retired particle **do**
    $i \leftarrow$ label of next edge in clockwise direction
  return i

4 Self-stabilizing Leader Election and Shape Formation

Consider the variant of the geometric amoebot model in which faults can occur that arbitrarily corrupt the local memory of a particle. Recall that for an algorithm to solve the leader election problem in a self-stabilizing manner, it has to satisfy the following requirements: First, from any initial system state (in which the particle structure is connected) the particle system eventually reaches a final system state while preserving connectivity, i.e., eventually a unique leader will be established. Second, once a final system state is reached, the system has to remain in that state as long as no faults occur. Analogous requirements have to be satisfied for self-stabilizing shape formation.

Our leader election algorithm can be extended to a self-stabilizing leader election algorithm with $O(\log^* n)$ memory using the results of [5, 29] (i.e., we use their self-stabilizing reset algorithm on every cycle in order to recover from

failure states). However, it is not possible to design a self-stabilizing algorithm for the line formation. The reason for this is that even a much simpler problem called *movement problem* cannot be solved in a self-stabilizing manner. It is easy to see that if the movement problem cannot be solved in a self-stabilizing manner, then also the line formation problem cannot be solved in a self-stabilizing manner.

In the movement problem we are given an initial distribution A of particles that can be in a contracted as well as expanded state, and the goal is to change the set of nodes occupied by the particles without causing disconnectivity. For the ring of expanded particles it holds that for any protocol P there is an initial state so that P does not solve the movement problem. To show this we consider two cases: suppose that there is any state s for some particle in the ring that would cause that particle to contract. In this case set two particles on opposite sides of the ring to that state, and the ring will break due to their contractions. Otherwise, P would not move any particle of the ring, so also in this case it would not solve the movement problem in a self-stabilizing manner.

5 Conclusion

We think that the algorithms presented for the geometric amoebot model can be extended for the case that G is a different regular grid graph embedded in the two-dimensional Euclidean plane. As future work, we would like to identify the minimum set of key geometric properties that G must have in order for the proposed algorithms to work. Also, if in the geometric amoebot model, the particles had a common sense of direction, we would like to investigate whether leader election could be solved deterministically using a slight modification of our algorithm: for each border the last remaining candidate is deterministically chosen to be the "east-most" particle of the set of the "south-most" particles. This algorithm would be similar to the one proposed in [26] for tile self-assembly systems.

References

1. L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(11):1021–1024, 1994.
2. C. Agathangelou, C. Georgiou, and M. Mavronicolas. A distributed algorithm for gathering many fat mobile robots in the plane. In *Proceedings of the 32nd ACM symposium on Principles of distributed computing (PODC)*, pages 250–259, 2013.
3. D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.
4. D. Arbuckle and A. Requicha. Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations. *Autonomous Robots*, 28(2):197–211, 2010.
5. B. Awerbuch and R. Ostrovsky. Memory-efficient and self-stabilizing network {RESET} (extended abstract). In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 254–263, 1994.

6. L. Barriere, P. Flocchini, E. Mesa-Barrameda, and N. Santoro. Uniform scattering of autonomous mobile robots in a grid. *Int. Journal of Foundations of Computer Science*, 22(3):679–697, 2011.
7. A. Bhattacharyya, M. Braverman, B. Chazelle, and H.L. Nguyen. On the convergence of the hegselmann-krause system. In *Proceedings of the 4th Innovations in Theoretical Computer Science (ITCS)*, pages 61–66, 2013.
8. D. Boneh, C. Dunworth, R. J. Lipton, and J. Sgall. On the computational power of DNA. *Discrete Applied Mathematics*, 71:79–94, 1996.
9. V. Bonifaci, K. Mehlhorn, and G. Varma. Physarum can compute shortest paths. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 233–240, 2012.
10. Z. J. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *International Journal of Robotics Research*, 23(9):919–937, 2004.
11. B. Chazelle. Natural algorithms. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 422–431, 2009.
12. H. Chen, D. Doty, D. Holden, C. Thachuk, D. Woods, and C. Yang. Fast algorithmic self-assembly of simple shapes using random agitation. In *Proceedings of the 20th International Conference on DNA Computing and Molecular Programming, (DNA)*, pages 20–36. Springer, 2014.
13. M. Chen, D. Xin, and D. Woods. Parallel computation using active self-assembly. In *Proceedings of the 19th International Conference on DNA Computing and Molecular Programming, (DNA)*, pages 16–30. Springer, 2013.
14. K. C. Cheung, E. D. Demaine, J. R. Bachrach, and S. Griffith. Programmable assembly with universally foldable strings (moteins). *IEEE Transactions on Robotics*, 27(4):718–729, 2011.
15. G. Chirikjian. Kinematics of a metamorphic robotic system. In *Proceedings of the 1994 International Conference on Robotics and Automation,(ICRA)*, pages 449–455, 1994.
16. M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.
17. R. Cohen and D. Peleg. Local spreading algorithms for autonomous robot systems. *Theoretical Computer Science*, 399(1-2):71–82, 2008.
18. S. Das, P. Flocchini, N. Santoro, and M. Yamashita. On the computational power of oblivious robots: forming a series of geometric patterns. In *Proceedings of 29th ACM Symposium on Principles of Distributed Computing (PODC)*, 2010.
19. X. Defago and S. Souissi. Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity. *Theoretical Computer Science*, 396(1-3):97–112, 2008.
20. E. D. Demaine, M. J. Patitz, R. T. Schweller, and S. M. Summers. Self-assembly of arbitrary shapes using rnase enzymes: Meeting the kolmogorov bound with small scale factor (extended abstract). In *Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science,*, pages 201–212, 2011.
21. Z. Derakhshandeh, S. Dolev, R. Gmyr, A. Richa, C. Scheideler, and T. Strothmann. Brief announcement: Amoebot — a new model for programmable matter. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures, (SPAA)*, pages 220–222, 2014.
22. D. Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55(12):78–88, 2012.

23. P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. *Algorithmica*, 65(3):562–583, 2013.
24. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1):412–447, 2008.
25. T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss. Self organizing robots based on cell structures - cebot. In *Proceedings of the International Conference on Intelligent Robots and Systems, (IROS)*, pages 145–150, 1988.
26. J. Hendricks, M. J. Patitz, and T. A. Rogers. Replication of arbitrary hole-free shapes via self-assembly with signal-passing tiles. *arXiv preprint arXiv:1503.01244*, 2015.
27. T.-R. Hsiang, E. Arkin, M. Bender, S. Fekete, and J. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Proceedings of the 5th Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 77–94, 2002.
28. A. Itai and M. Rodeh. Symmetry breaking in distributive networks. In *22nd Annual Symposium on Foundations of Computer Science, (FOCS)*, pages 150–158, 1981.
29. G. Itkis and L. Levin. Fast and lean self-stabilizing asynchronous protocols. In *35th Annual Symposium on Foundations of Computer Science,(FOCS)*, pages 226–239, 1994.
30. S. Kernbach, editor. *Handbook of Collective Robotics – Fundamentals and Challanges*. Pan Stanford Publishing, 2012.
31. P. Kling and F. Meyer auf der Heide. Convergence of local communication chain strategies via linear transformations. In *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures, (SPAA)*, pages 159–166, 2011.
32. K. Li, K. Thomas, C. Torres, L. Rossi, and C.-C. Shen. Slime mold inspired path formation protocol for wireless sensor networks. In *Proceedings of the 7th International Conference on Swarm Intelligence, (ANTS)*, pages 299–311, 2010.
33. J. McLurkin. *Analysis and Implementation of Distributed Algorithms for Multi-Robot Systems*. PhD thesis, Massachusetts Institute of Technology, 2008.
34. M. J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.
35. M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.
36. J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17(2):171–189, 2004.
37. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional dna crystals. *Nature*, 394(6693):539–544, 1998.
38. D. Woods. Intrinsic universality and the computational power of self-assembly. In Turlough Neary and Matthew Cook, editors, *Proceedings of the 6th Conference on Machines, Computations and Universality 2013, (MCU)*, volume 128 of *EPTCS*, pages 16–22, 2013.
39. D. Woods, H. Chen, Goodfriend. S., N. Dabby, E. Winfree, and P. Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science, ITCS*, pages 353–354, 2013.
40. M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robotics Automation Magazine*, 14(1):43–52, 2007.