Brief Announcement: On the Feasibility of Leader Election and Shape Formation with Self-Organizing Programmable Matter*

Zahra Derakhshandeh†
Arizona State University, USA
zderakhs@asu.edu

Robert Gmyr‡
University of Paderborn, Germany
gmyr@mail.upb.de

Thim Strothmann‡
University of Paderborn, Germany
thim@mail.upb.de

Rida Bazzi
Arizona State University, USA
bazzi@asu.edu

Andréa W. Richa†
Arizona State University, USA
aricha@asu.edu

Christian Scheideler‡
University of Paderborn, Germany
scheideler@.upb.de

ABSTRACT

Imagine that we had a piece of matter that can change its physical properties like shape, density, conductivity, or color in a programmable fashion based on either user input or autonomous sensing. This is the vision behind what is commonly known as *programmable matter*. Many proposals have already been made for realizing programmable matter, ranging from DNA tiles, shape-changing molecules, and cells created via synthetic biology to reconfigurable modular robotics. We are particularly interested in programmable matter consisting of simple elements called *particles* that can compute, bond, and move, and the feasibility of solving fundamental problems relevant for programmable matter with these particles. As a model for that programmable matter, we will use a general form of the amoebot model first proposed in SPAA 2014, and as examples of fundamental problems we will focus on leader election and shape formation. For shape formation, we investigate the line formation problem, i.e. we are searching for a local-control protocol so that for any connected structure of particles, the particles will eventually form a line.

Prior results on leader election imply that in the general amoebot model there are instances in which leader election cannot be solved by local-control protocols. Additionally, we can show that if there is a local-control protocol that solves the line formation problem, then there is also a protocol that solves the leader election problem, which implies that in the general amoebot model also the line formation problem cannot be solved by a local-control protocol.

We also consider a geometric variant of the amoebot model by restricting the particle structures to form a connected subset on a triangular grid. For these structures we can show that there are local-control protocols for the leader election problem and the line formation problem. The protocols can also be adapted to other regular geometric structures demonstrating that it is advisable to restrict particle structures to such structures.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General

1. INTRODUCTION

There have been many conceptions of programmable matter, and thus many avenues of research using that name, each pursuing solutions for specific application scenarios with their own, special capabilities and constraints. However, most of the research in this area has been done by scientists from the natural sciences and the robotics area and not so much from theoretical computer science, so not much is known yet about which assumptions or primitives allow the development of distributed algorithms that can solve relevant problems for programmable matter in a self-organizing way. Notable exceptions are DNA tile assembly and population protocols. However, most of these approaches are passive in one or more aspects or require powerful entities while we are interested in studying the feasibility of solving problems with simple entities that can not just compute but also control their movements. For that we use a general form of the amoebot model first proposed in [2].

1.1 The general amoebot model

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

As a base for our amoebot 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

*A full paper presenting the results outlined in this brief announcement will appear at the 21st International Conference on DNA Computing and Molecular Programming (DNA21).

†This work was supported in part by the NSF under Awards CCF-1353089 and CCF-1422603.

‡Supported in part by DFG grant SCHE 1592/3-1

PODC'15, July 21–23, 2015, Donostia-San Sebastián, Spain.

http://dx.doi.org/10.1145/2767386.2767451.

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 equals all possible transitions between positions.

Each particle occupies either a single node or a pair of adjacent nodes in G, 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 in which it can store some bounded amount of information, and any pair of connected particles has a bounded shared memory that can be read and written by both of them and that can be accessed using the edge label associated with that connection.

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 — in the latter case, it also knows along which edge it is 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 (resembling loosely the movement of amoeba), they can also be interpreted as a particle "looking ahead" and establishing new connections (by expanding) before it fully moves to a new position and severs the old connections it had (by contracting). In Figure 1, we illustrate a set of particles (some contracted, some expanded) using the infinite regular triangular grid graph G_{eqt} as the underlying graph G .

Figure 1: The left part shows an example of a particle structure in G_{eqt}. 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 solid and the two inner borders are dashed.

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 arbitrary bounded amount of computation (involving its local memory as well as the shared memories with its neighboring particles) followed by no or a single movement. A *round* is over once every particle has been activated at least once.

2. LEADER ELECTION AND LINE FORMATION IN THE GENERAL AMOEBOT MODEL

Here we focus on the feasibility of leader election and shape formation for particles with constant memory. For the shape formation, we just focus on forming a line of particles, that is, we are searching for a local-control algorithm such that for any initial set $A \subseteq V$ of positions occupied by particles where $G|_A$ (i.e., the subgraph of G induced by A) is connected, the particles will eventually rearrange themselves into a line without losing connectivity. Given that we do not assume any underlying geometric information regarding G, by forming a line, we mean that for the final set of occupied positions A', $G|_{A'}$ forms an arbitrary simple path.

Suppose that there is a protocol for the line formation problem in the general amoebot model. Then it is easy to extend it to a leader election protocol: once the line has been formed, its two endpoints contend for leadership using tokens with random bits sent back and forth. On the other hand, one can deduce from [3] that in the general amoebot model there is no local-control protocol for leader election. Hence, also line formation cannot be solved. So additional assumptions are needed to solve these.

3. LEADER ELECTION IN THE GEOMETRIC AMOEBOT MODEL

In this section we assume that G is equal to G_{eqt} (see Figure 1) and that the edges are labeled in a consecutive way in clockwise direction so that every particle has the same sense of clockwise direction, but the labelings do not have to be consistent. So the particles may not have a common sense of orientation. However, our model allows a particle to maintain a fixed orientation as it moves because in an expanded state it knows along which edge it expands from both sides.

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 common sense of clockwise rotation can be used to give each border a unique orientation.

The leader election algorithm operates independently on each border. Due to the nature of G_{eqt} a particle can be adjacent to at most three different empty regions for each region a particle simulates one agent. For simplicity and ease of presentation we assume for now that agents have a global view of the border they are part of and that agents act synchronously. At any given time, some subset of agents on a border will consider themselves *candidates*, i.e. potential leaders of the system. Initially, every agent considers itself a candidate. Between any two candidates on a border there is a (possibly empty) sequence of non-candidate agents. We call such a sequence a *segment* and specifically refer to the segment coming after (before) a candidate c in the direction of the border as the *front segment* (*back segment*) of c. We denote the lengths of those segments as $|fs(c)|$ and $|bs(c)|$. We use *front candidate* ($fc(c)$) and *back candidate* ($bc(c)$) to denote the candidates at the end of these segments. 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 the direction of the border. We say a candidate c_1 *covers* a candidate c_2 (or c_2 *is covered by* c_1) if $|fs(c_1)| > d(c_2, c_1)$. The leader election 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.

Algorithm 1 Leader Election in the Geometric Amoebot Model

```
Subphase 1:
pos ← position of fc
if covered by any candidate or |fs| < |bs| 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
```

In the first subphase candidates are eliminated deterministically. In the second subphase candidates that survived the first subphase are eliminated in a randomized fashion by transferring candidacy to another agent. This transferral of candidacy means that c withdraws its own candidacy but at the same time promotes the agent at position pos (i.e., the front candidate of c in subphase 1) to be a candidate. Subphase 1 and 2 make sure that eventually there is only one leader on each border. Since we assume that each particle has a global view, it can detect whether it is on the outer or an inner border. Hence, the algorithm will elect a unique leader.

Algorithm 1 can be implemented entirely as a local-control protocol. This implementation heavily relies on token passing. Since a detailed description of the protocol cannot be presented in this brief announcement, we focus on the local-control protocol for one part of the algorithm: the test whether the last remaining candidate of a border is on the outer border or an inner border. We can achieve this as follows. The candidate sends a token along its border that keeps track of the sum of angles the path takes with respect to the orientation of the candidate. Once the token manages to return to its origin (the candidate itself), the sum of angles stored in the token is inspected. It turns out that it is always 360° for the outer border and -360° for an inner border. Since we work on G_{eqt}, we just need to count integer multiples of 60° so that instead of the actual angle we can simply store an integer k in the token such that the angle is $k \cdot 60^\circ$. Note that the range of k during the traversal of the path is unbounded. However, since we are only interested in whether the value of k is 6 or -6 once the token returns to its origin, we can use a counter modulo 5 so that for the outer border we get $k = 1$ and for an inner border $k = 4$. This way, the token only needs 3 bits of memory independently of the shape of the border.

4. LINE FORMATION IN THE GEOMETRIC AMOEBOT MODEL

Our line formation algorithm is based on the leader election strategy above. Once a leader has been determined, the leader is used as a seed to arrange the particles in a straight line starting from that leader. Every time the line can be extended by a particle, that particle simply joins the line. Whenever the line cannot be extended any more because of an empty spot, particles are drawn to that spot using the following idea:

All particles connected to the line that are not yet part of it, and have an empty spot in the direction in which the line needs to grow, form the root of a tree of particles so that all particles belong to one of these trees. Every root tries to travel along the line in the direction in which it is grown. If this is not possible any more because the next spot is already occupied by an particle, it joins the tree of that particle. Otherwise, it moves forward on top of the line and drags the particles of its tree with it in a worm-like fashion (which can be realized with the handover operation). This guarantees that eventually all particles end up being part of the line.

5. SELF-STABILIZING ALGORITHMS

The leader election algorithm for G_{eqt} can be extended to a self-stabilizing leader election algorithm with $O(\log^* n)$ memory using the results of [1, 4] (i.e., we use their self-stabilizing reset algorithm on every border in order to recover from failure states). However, it is not possible to design a self-stabilizing algorithm for the line formation problem. The reason for this is that even a much simpler problem we call the *movement 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 essentially 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. 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. Hence, while self-stabilizing leader election is possible, self-stabilizing line formation is not possible, which also implies that it is not possible for the particles to *decide* in this setting when a leader has been successfully chosen.

6. REFERENCES

[1] Baruch Awerbuch and Rafail Ostrovsky. Memory-efficient and self-stabilizing network RESET (extended abstract). In *13th Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17, 1994*, pages 254–263, 1994.

[2] Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: amoebot - a new model for programmable matter. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, Prague, Czech Republic - June 23 - 25, 2014*, pages 220–222, 2014.

[3] Alon Itai and Michael Rodeh. Symmetry breaking in distributive networks. In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 150–158, 1981.

[4] Gene Itkis and Leonid A. Levin. Fast and lean self-stabilizing asynchronous protocols. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 226–239, 1994.