Algorithmic Foundations of Self-Organizing Programmable Matter

by

Zahra Derakhshandeh

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2017 by the
Graduate Supervisory Committee:

Andrea Richa, Co-Chair
Arunabha Sen, Co-Chair
Guoliang Xue
Christian Scheideler

ARIZONA STATE UNIVERSITY

August 2017

ABSTRACT

Imagine that we have a piece of matter that can change its physical properties like its 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*. Envisioning systems of nano-sensors devices, programmable matter consists of systems of simple computational elements, called *particles*, that can establish and release bonds, compute, and can actively move in a self-organized way. In this dissertation the feasibility of solving fundamental problems relevant for programmable matter is investigated. As a model for such self-organizing particle systems (SOPS), the *geometric amoebot* model is introduced. In this model, particles only have local information and have modest computational power. They achieve locomotion by expanding and contracting, which resembles the behavior of amoeba. Under this model, efficient local-control algorithms for the leader election problem in SOPS are presented. As a central problem for programmable matter, shape formation problems are then studied. The limitations of solving the leader election problem and the shape formation problem on a more general version of the amoebot model are also discussed. The "smart paint" problem is also studied which aims at having the particles self-organize in order to uniformly coat the surface of an object of arbitrary shape and size, forming multiple coating layers if necessary. A Universal Coating algorithm is presented and shown to be asymptotically worst-case optimal both in terms of time with high probability and work. In particular, the algorithm always terminates within a *linear number of rounds* with high probability. A *linear lower bound on the competitive gap* between fully local coating algorithms and coating algorithms that rely on global information is presented, which implies that the proposed algorithm is also optimal in a competitive sense. Simulation results show that the competitive ratio of the proposed algorithm may be better than linear in practice.

Developed algorithms utilize only local control, require only constant-size memory particles, and are asymptotically optimal in terms of the total number of particle movements needed to reach the desired shape configuration.

DEDICATION

I dedicate my dissertation to my beloved family, my parents, Hassan Derakhshandeh and Fakhri Shafiei, my brother, Behnood Derakhshandeh, and my husband, Babak Esmaeili, for your unconditional love and support.

My special thanks go to the dearest husband, Babak, for supporting me all the way! I am truly thankful for always being there for me. Without your endless help, support and encouragement it simply never would have been.

I also dedicate my success to my angel baby, Aniya, for taking a part of this journey with her mommy.

ACKNOWLEDGMENTS

First of all, I would like to express my deepest appreciation and gratitude to my supervisor, Prof. Andrea Richa, for her professional guidance, patience, continuous support, motivation, and encouragement over my Ph.D. study and research. I could not have imagined having a better mentor for my Ph.D. study. I greatly appreciate her insightful discussions and invaluable guidance in my research.

My sincere thanks also goes to all my committee members, Prof. Guoliang Xue, Prof. Arunahba Sen, Dr. Theodore Pavlic, and specifically Prof. Christian Scheideler for their invaluable advices.

I would also like to thank Araxi Hovhannessian, Christina Sebring, Monica Dugan, Pamela Dunn, and Brint Macmillan for always being helpful and positive.

I also thank my colleagues, labmates, and friends: Xinhui Hu, Mengxue Liu, Chenyang Zhou, Robert Gmyr, Thim Strothmann, and Joshua Daymude for the pleasant and inspiring discussion. Their friendship is a valuable experience for me.

Last but not least, I would like to thank my husband Babak and my parents who give me endless love and constant encouragement during my study at ASU and throughout my life.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

Recent advances in microfabrication and cellular engineering foreshadow that in the next few decades it might be possible to assemble simple information processing units at almost no cost. Myriads of these small-scale units could be combined to powerful systems capable of solving intricate tasks. This vision of building cheap microscopic processing units is supported by the progress made in manufacturing microelectronic mechanical components, such that one can anticipate integrating logic circuits, microsensors, actuators, and communications devices on the same chip. Imagine coating bridges and buildings with smart paint that senses and reports on traffic and wind loads and monitors structural integrity. A smart-paint coating on a wall could sense vibrations, monitor the premises for intruders, and cancel noise. There has also been amazing progress in understanding the biochemical mechanisms in individual cells such as the mechanisms behind cell signaling and cell movement [4]. Recent results have also demonstrated that, in principle, biological cells can be turned into finite automata [10], so one can imagine that some day one can tailor-make biological cells to function as sensors and actuators, as programmable delivery devices, and as chemical factories for the assembly of nano-scale structures. Particularly interesting applications for this technology would be the construction of molecular-scale electronic structures and of nano-scale devices for surgery as well as cancer treatment [58].

One can envision producing vast quantities of individual microscopic computational particles —whether microfabricated particles or engineered cells—to form *programmable matter*, as coined by Toffoli and Margolous [55]. These particles are pos-

sibly faulty, sensitive to the environment, and may produce various types of local actions that range from changing their internal state to communicating with other particles, sensing the environment, moving to a different location, changing shape or color, or even replicating. Those individual local actions may then be used to change the physical properties, color, and shape of the matter at a global scale.

Envisioning systems of nano-sensors devices, we are particularly interested in programmable matter consisting of systems of simple computational particles that can establish and release bonds, can compute, 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 (SOPS), we propose *amoebot*, a new amoeba-inspired model for programmable matter. In our model, the programmable matter consists of particles that can bond to neighboring particles and use these bonds to form connected structures. Particles only have local information and have modest computational power: Each particle has only a constant-size memory and behaves similarly to a finite state machine. The particles act asynchronously and they achieve locomotion by expanding and contracting, which resembles the behavior of amoeba [4].

There are many fundamental problems that one could study for such systems. In this dissertation, we will focus on the leader election problem, shape formation problems, and coating problems, always aiming at obtaining efficient algorithms for those. In the leader election problem, we are given a set of particles and we would like to select one of these as the leader. In shape formation problems, one would like the system of particles to assume a specific shape (e.g., a disc, or a line, or even an arbitrary convex shape). We will also consider coating problems, which is to uniformly coat a given arbitrary object by a layer (possibly layers) of particles.

This dissertation is organized as follows: In Section 1.1, we study the related work. In Section 2, we propose the amoebot model and study the model in two different settings, the general setting and the geometric setting. In Section 3, based on the geometric model, we present efficient local-control algorithm for leader election problem requiring only particles with constant-size memory. In Section 4, we propose an algorithm for line formation problem in geometric amoebot model. In addition, we present a general algorithmic framework for more complex shape formation problems in SOPS, and show direct applications of this framework to the problems of having the particle system self-organize to form a hexagonal or triangular shape. We analyze our proposed algorithms in terms of amount of work required to get the desired shape and we analyze the correctness of the algorithms. Then, we discuss the limitations of solving leader election and line formation problems within the general amoebot model, in Section 5. We present a *worst-case work-optimal Universal Coating algorithm* that uniformly coats any object of arbitrary shape and size that allows a uniform coating in Section 6. In Section 7, we then continue the study of our Universal Coating algorithm by focusing on its runtime analysis, showing that our algorithm terminates within a *linear number of rounds* with high probability. We conclude the work in Section 8.

1.1 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 [2, 14, 21, 29, 60], tile

self-assembly systems in general (e.g., see the surveys in [37, 50, 61]), population protocols [5], and slime molds [15, 46]. 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., [38]), gathering problems (e.g., [3, 23]), shape formation problems (e.g., [39, 52]), and to understand the global effects of local behavior in natural swarms like social insects, birds, or fish (see e.g., [11, 18]). Surveys of recent results in swarm robotics can be found in [43, 48]; other samples of representative work can be found in e.g., [6, 8, 24, 25, 28, 41, 44]. Coating of objects is commonly not studied as a stand-alone problem, but is part of *collective transport* (e.g., [59]) or *collective perception* (e.g., see respective section of [16]). Some research focuses on coating objects as an independent task under the name of *target surrounding* or *boundary coverage*. The techniques used in this context include stochastic robot behaviors [45, 51], rule-based control mechanisms [12] and potential field-based approaches [13]. 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.

In a recent paper [49], Michail and Spirakis propose a model for network construction that is inspired by population protocols [5]. The population protocol model relates to self-organizing particles systems, but is also intrinsically different: agents

(which would correspond to our particles) freely move in space and can establish connections to any other agent in the system at any point in time, following the respective probabilistic distribution. In the paper the authors focus on network construction for specific topologies (e.g., spanning line, spanning star, etc.). However, in principle, it would be possible to adapt their approach also for studying coating problems in SOPS.

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., [40, 63]). *Metamorphic robots* form a subclass of self-reconfigurable robots that share some of the characteristics of our geometric model [22]. The hardware development in the field of self-reconfigurable robotics has been complemented by a number of algorithmic advances (e.g., [17, 52, 57]), 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 [19, 20, 62] 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.

Our proposed model has many direct applications in the context of nano-sensor networks or modular nano-robotics, and may even extend beyond those: For example

in scratch-wound healing, epithelial cells as our skin try to cover the gap caused by a scratch while they stick together and move as a group in an analogous fashion to particles moving in our model [56].

Chapter 2

MODEL

In this section we propose a new amoeba-inspired model called *amoebot* model for programmable matter. We study two variations of this model throughout this work. Firstly, we consider a general version of our model in Section 2.1. Then, in Section 2.2, we consider a geometric variant of the general model.

2.1 General 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 [4].

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 a particle can perform in one action as well as all places where neighboring particles can bond to each other.

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* by a *bond*, 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. Each particle has a constant-size shared/local memory.

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. 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 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).

The *configuration* C of the system at the beginning of time t consists of the nodes in G_{eqt} occupied by the object and the set of particles, and for each particle p, C contains the current state of p, including whether it is expanded or contracted, its port labeling, and the contents of its local memory. We say a particle system in a system configuration is connected if it is in a configuration where the graph $G|_A$ induced by the set of nodes $A \subseteq V$ occupied by particles is connected.

Following the standard asynchronous model of computation [47], we assume that the system progresses through atomic *activations* of individual particles [1]. At each (atomic) activation a particle can perform at most one movement and an arbitrary bounded amount of computation, involving its local memory and the shared memories of its neighbors. A classical result under this model is that for any asynchronous execution of atomic particle activations, we can organize these activations sequentially still producing the same end configuration [47]. We count (asynchronous) time in terms of the number of activations. A *round* is over once each particle has been activated at least once. We assume the activation sequence to be *fair*, i.e., for each particle p and any point in time t, p will eventually be activated at some time $t' \geq t$.

Hence, a *computation* of a particle system is a potentially infinite sequence of actions $A_1, A_2, \ldots$ based on some initial system configuration C_0, where action A_i transforms system configuration C_{i-1} into system configuration C_i. While the (parallel) time complexity of a computation is usually measured in rounds, the *work* spent by the particles till time t is measured by the number of movements they have done until that point (We ignore other state changes since their energy consumption should be irrelevant compared to the energy for a movement.).

[1] In reality, this may not be the case, but a local contention resolution rule could be used to make sure that only one particle within a local neighborhood is active at a time, which would be sufficient for our results to hold in a concurrent environment.

Let $\mathcal{C}$ be the set of all system configurations with n particles in which the particle system is connected. In general, a *computational problem* P for the particle system is specified by a set $\mathcal{C}' \subseteq \mathcal{C}$ of permitted initial system configurations with n particles and a mapping $F : \mathcal{C}' \to 2^{\mathcal{C}}$, where $F(c) \subseteq \mathcal{C}$ determines the set of permitted *final* configurations for any initial configuration $c \in \mathcal{C}'$. A particle system *solves* problem $P = (\mathcal{C}', F)$ if for any initial system configuration $c \in \mathcal{C}'$, all computations of the particle system eventually reach a system configuration in $F(c)$ without losing connectivity, and whenever such a system configuration is reached for the first time, the system stays in $F(c)$. If for all computations a final configuration 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 configuration does not necessarily mean that all particles decided to halt. If $\mathcal{C}' = \mathcal{C}$, so *any* initial configuration is permitted (including arbitrary faulty configurations, as long as the particle system is connected), then a particle system solving P is also said to be *self-stabilizing*.

2.2 Geometric Amoebot Model

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} [2]. See Figure 2.1.

[2]The triangular grid graph G_{eqt} is the dual graph of a regular hexagonal tiling in 2D space.

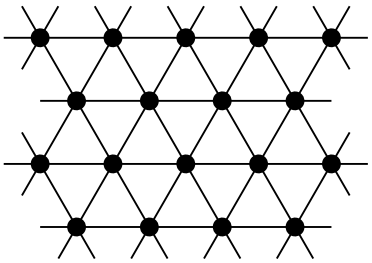

Figure 2.1: Figure shows a section of G_{eqt}; nodes of G_{eqt} are shown as black circles.

As before, particles are anonymous but the bonds of each particle have unique labels. Therefore, a particle can uniquely identify each of its outgoing edges. We also assume that the particles have a common *chirality*, i.e., they all have the same notion of *clockwise direction*, which allows each particle p to order its head bond labels. W.l.o.g.[3], we assume that each particle labels its head and tail bonds from 0 to 5 in clockwise order. However, particles do not have a common sense of orientation since they can have different offsets of the labelings (Figure 2.2(b)). In Figure 2.2(a), we illustrate a set of particles (some contracted, some expanded) on the underlying graph G_{eqt}.

Figure 2.3 illustrates an example of particles p and q executing a handover in G_{eqt}. As the result of executing the handover, the expanded particle p contracts and the contracted particle q expands.

[3]Without loss of generality.

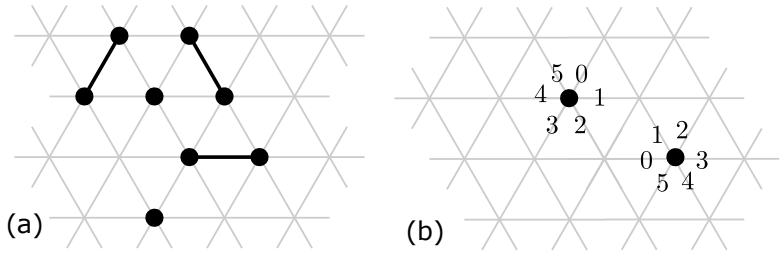

Figure 2.2: (a) shows an example of a particle structure in the geometric amoebot model consisting of five particles on G_{eqt}: the underlying graph G_{eqt} is shown as a gray mesh; a particle occupying a single node, a contracted particle, is depicted as a black circle, and a particle occupying two nodes, an expanded particle, is depicted as two black circles connected by an edge. (b) depicts two particles occupying two non-adjacent positions on G_{eqt}; the particles have different offsets for their head bond labelings.

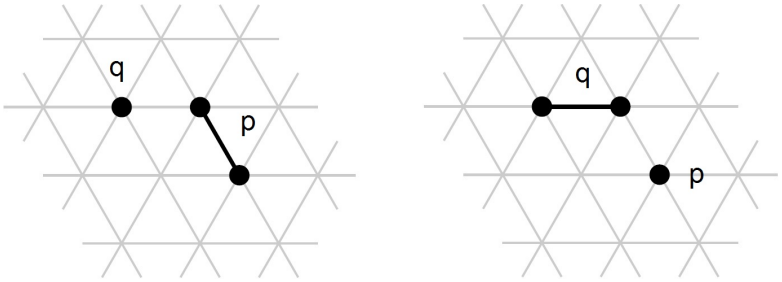

Figure 2.3: Two particles performing a handover. The left part shows an expanded particle p and a contracted particle q before executing the handover. The right part depicts the configuration of the particles after performing a handover. This ends up p to contract and q to expand.

Chapter 3

LEADER ELECTION IN SOPS

3.1 Introduction

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 majority problem (all particles have to agree on the value held by the majority) 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. Based on the geometric model, we present efficient local-control algorithm for leader election problem in SOPS. For the leader election problem, initially we assume that the particle system is connected and all memories are empty. The set of final (goal) system configurations, contains any configuration 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. Throughout this section, we assume for the sake of simplicity that in an initial configuration all particles are contracted. Our algorithm can easily be extended to dispose of this assumption.

3.2 Our Contributions

Here, we focus on the problem of solving leader election for particles with constant memory. 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 boundary between the particle structure and an empty region (inside or outside of it) in G_{eqt}. Our leader election algorithm assumes that the system starts in a well-initialized configuration. We also study the leader election problem in the geometric amoebot model assuming that all the particles have a common *orientation* and present a deterministic algorithm for that which needs linear number of the rounds with respect to the number of particles in the system to terminate. In Section 5, we will show that under the general amoebot model, the leader election problem is not decidable by any distributed algorithm.

3.3 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 3.4 we present a high level descriptions of the techniques used in the local-control protocol, which relies heavily on token passing.

3.3.1 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 *boundary*. 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 boundary corresponding to the infinite empty region to be the unique *outer boundary* and refer to a boundary that corresponds to a finite empty region as an *inner boundary*, see Figure 3.1.

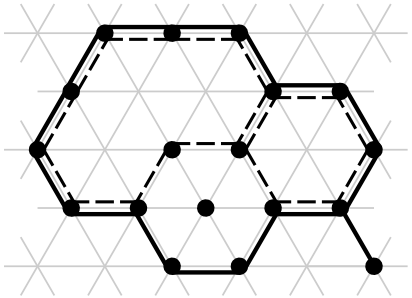

Figure 3.1: The figure shows a particle structure with 3 boundaries. The outer boundary is shown as a solid line and the two inner boundaries are shown as dashed lines.

The particles occupying a boundary can instantly (i.e., without communication) organize themselves into a cycle using only local information: Consider a boundary corresponding to an empty region R. Let p be a particle occupying a node v of the boundary. By definition there exists a non-occupied node $w \in R$ that is 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 boundary. 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 boundaries 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 3.2. Observe that from a global perspective the cycle of the outer boundary is oriented clockwise while a cycle of an inner boundary is oriented counter-clockwise. This is a direct consequence of the way the predecessors and successors of an agent are defined.

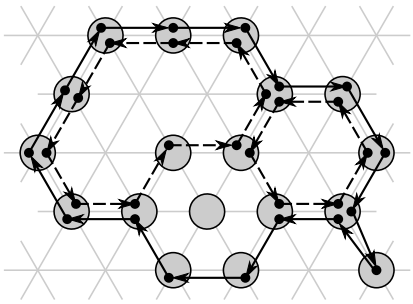

Figure 3.2: The depicted particle system is the same as in the Figure 3.1. In this figure particles are depicted as gray circles. The black dots inside of a particle represent its agents. As in Figure 3.1 the outer boundary is solid and the two inner boundarys are dashed.

3.3.2 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. poten-

tial 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 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 3.3). 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 3.3).

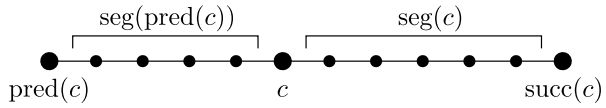

Figure 3.3: A cutout from a cycle that is oriented to the right. Non-candidate agents are small black dots, candidates are bigger dots. The candidate c covers $pred(c)$ since $|seg(c)| > d(pred(c), c)$.

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.

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:

$pos \leftarrow$ 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 boundary **then**

    if outside boundary **then**

        return leader

    else

        return not leader

As we show in [36, 35], Algorithm 1 decides the leader election problem in the geometric amoebot model and requires $O(L_{\max})$ rounds on expectation, where $L_{\max}$ is the length of the longest cycle in the particle system.

3.4 Asynchronous Local-Control Protocol

Here we present a high level description on how specific parts of the algorithm can be realized as an asynchronous local-control protocol (See [26] for proofs). We assume that tokens are messages of constant size. The tokens of each subphase of our algorithm are independent of each other, so an agent has to handle distinct tokens

for each phase at the same time. If not otherwise specified, we assume that tokens of a single subphase move through the agents in a well-ordered manner (i.e., a token does not surpass another token in front of it but waits until the agent is free to hold it).

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. The candidate inspects all non-leader agents on its segment. 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. By a simple geometrical argument it is clear that in case the segment is the whole cycle, the vectors cancel out component wise (see Figure 3.4). 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.

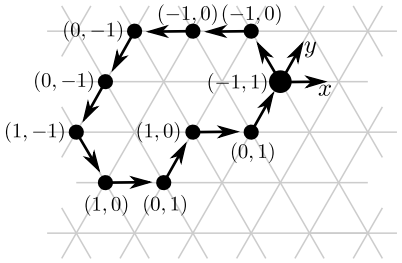

Figure 3.4: An example of solitude verification: the candidate (shown slighly bigger) has enforced its orientation (x and y arrows) on all agents. All non-candidate agents determined the offset to the succeeding agent in direction of the cycle (arrows and numbers at nodes).

Inner Outer Boundary Test: The distinction between inner boundaries and outer boundary is readily apparent when looking at a particle system from a global perspective. Here we are interested in whether the last candidate is able to locally distinguish between the two. The last candidate of a cycle can decide whether its cycle corresponds to an inner or the outer boundary by using the geometrical properties induced by G_{eqt} together with the ability to distinguish clockwise from counter-clockwise rotation as follows. A cycle corresponding to an inner boundary has counter-clockwise rotation while a cycle corresponding to the outer boundary has clockwise rotation, see Figure 3.2. The candidate sends a token along the cycle that sums the angles of the turns the cycle takes, see Figure 3.5.

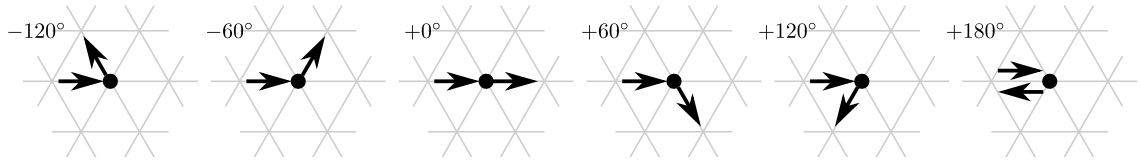

Figure 3.5: The angle between the directions a token enters and exits an agent.

When the token returns to the candidate its value represents the external angle of the polygon corresponding to the cycle while respecting the rotation of the cycle. So it is $-360°$ for an inner boundary and $360°$ for the outer boundary. 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 (since for the outer boundary we get $k = 1$ and for an inner boundary we get $k = 4$).

Candidate Elimination: In subphase 1 of Algorithm 1 candidates use differences in length of their segments to eliminate other candidates. To implement this behavior in a local-control protocol, a candidate will virtually push its segment into the segment

of its preceding candidate and eliminate all candidates that are covered. If a candidate detects that its segment is too short to cover any candidate it revokes candidacy.

Consider a candidate c at the beginning of subphase 1. It sends a *starting token* along its segment (i.e., in direction of the cycle). Each non-candidate agent that receives the token forwards it and sends a *cover token* in opposing direction of the cycle. Candidate $succ(c)$ consumes the starting token and sends a special end of segment (EOS) cover token. As long as the cover tokens stay on the segment of c we consider them to be *passive*, i.e., they are only forwarded by non-candidate agents in a pipelined fashion. Once a token passes c on its path it becomes *active*. An active cover token matches with the first agent on its path that has not been matched by another active token of c.

Once an agent is matched it deletes all other tokens it holds (e.g., cover tokens from another candidate), except for passive tokens. If a candidate is matched it revokes candidacy immediately. Additionally, if it is in subphase 1 it consumes its own passive cover tokens instead of forwarding them (see Figure 3.6 for examples). It also clears its active tokens which implicitly unmatches all the agents that has been matched with its own active tokens. If the candidate is in subphase 2 or 3, no additional work has to be done (except for the aforementioned annulment). Once the EOS token has matched, the last matched agent sends a *finish token* back to c. As soon as c receives this token, subphase 1 is complete. If the cover tokens of c have not matched with another candidate (which can be detected by the finish token) c revokes candidacy and annuls all tokens sent out in subphase 1 (i.e., it unmatches all agents), otherwise the agents stay matched for subphase 2. Note, that in case the EOS token has matched with another candidate (i.e., both segments have equal length), both candidates stay candidates.

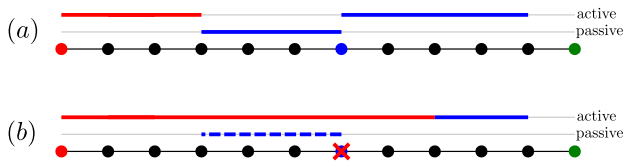

Figure 3.6: Two examples for the token-agent-matching process. Agents are nodes on the lowest line, which is part of a cycle directed to the right. Colored lines depict unmatched cover tokens occupying consecutive agents (agents below token lines are already matched). Tokens on the lower lane are still passive, tokens in the upper lane are active. **Case (a):** The red and blue candidate have already matched parts of their respective $seg(pred)$ segments. Passive tokens of blue are not canceled out by active tokens of red. **Case (b):** The red tokens have already matched up to the blue candidate and beyond. Blue revoked leadership, the passive tokens of blue (dotted lines) cannot become active and are simply consumed by the blue candidate.

Candidate Transferral: The candidate transfers candidacy to the agent at position pos (i.e., $succ(c)$ before subphase 1) with probability $\frac{1}{2}$ in subphase 2. In order to do so in a local-control protocol, the candidate uses the matched agents in from subphase 1. By using a token passing scheme similar to the first subphase, the matched agents from the first subphase create cover tokens that match with the agents on c's segment. Candidacy is transferred onward on the newly matched agents (which unmatches the agents implicitly) until position pos.

3.5 Deterministic Leader Elcetion

We now study the leader election problem in the geometric amoebot model assuming that all the particles have a common *orientation*. The orientation of a particle p whose head occupies node v is given by the direction of the vector (v, u), where u is the node adjacent to v via p's 0-labeled edge. Particles have no other global information. We describe the algorithm in the following section.

3.5.1 Algorithm

Here, similar to the randomized version of the leader election problem presented above, our approach organizes the particle system into a set of cycles and executes the algorithm on each cycle independently. Organization into cycles follows the rules we already mentioned in Section 3.3.1. Initially, every agent considers itself a candidate. As before, between any two candidates on a cycle there is a (possibly empty) sequence of non-candidate agents called a segment. For a candidate c we refer to the segment coming after c in the direction of the cycle as $seg(c)$. Recall that we refer to the candidate coming after c in the direction of the cycle as the succeeding candidate ($succ(c)$), and to the candidate coming before c as the preceding candidate ($pred(c)$). The leader election progresses in phases. In each phase, each candidate executes Algorithm 2. A phase consists of two subphases. As before, for any candidate c executing the algorithm, 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.

In subphase 1 of Algorithm 2, each candidate c on a cycle of boundary particles competes with $succ(c)$. Every agent in $seg(c)$ determines the offset to the succeeding agent in direction of the cycle. These offsets can be seen as vectors in two dimensional

plane. By using a token passing mechanism, particles try to match their vectors component-wise with another vector with an opposite component. The candidate c then inspects its segment. If all vectors are completely matched, it means c is the only candidate on that boundary (this corresponds to our solitude verification procedure we presented in randomized leader election algorithm, Section 3.4). Otherwise, if there are remaining negative (resp. positive) y-components unmatched, c is further north (resp. south) than $succ(c)$ and we say c *covers* (resp. is *covered* by) $succ(c)$. If all y-components of the vectors are matched, and there are remaining negative (resp. positive) x-components unmatched, c is further east (resp. west) than $succ(c)$, and again we say c *covers* (resp. is *covered* by) $succ(c)$. See Figure 3.7.

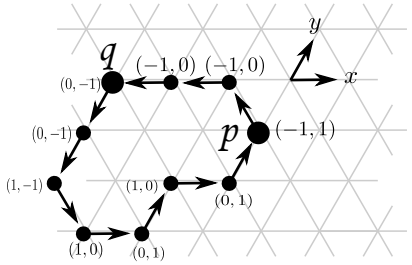

Figure 3.7: Two candidates p and q (shown slightly bigger) are shown in the figure. Non-candidate agents are small black dots. Common orientation for all the particles is given (x and y arrows). Each agent has obtained its offset to the succeeding agent (arrows and numbers at nodes). p inspects its segment. The unmatched y-component has positive sign, which means p is further south than q. Thus, p revokes its candidacy, and we say p is *covered by* q (q *covers* p).

Executing the algorithm by all particles, eventually the last remaining candidate on each cycle is deterministically chosen to be the "east-most" particle of the set of the "north-most" particles on that cycle. Assuming that c is the only candidate on the boundary, in subphase 2 of Algorithm 2, c detects if the boundary it belongs to

is an inner boundary or an outer boundary. If c is located on the outer boundary, it becomes the leader and broadcasts this information such that all particles can halt. The process of detecting inner or outer boundaries for a candidate c is exactly as we presented in Section 3.4.

Algorithm 2 Leader Election for a candidate c

Subphase 1:

if covered by a candidate **then**

    return not leader

Subphase 2:

if only candidate on boundary **then**

    if outside boundary **then** ▷ inner/outer boundary detection

        return leader

    else

        return not leader

3.5.2 Correctness of the Deterministic Leader Election Algorithm

In order to prove the correctness of our algorithm, we show that it satisfies the following conditions relating 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 candidacy.

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

Proof. We will show by induction that on the outer boundary's cycle there will always be at least one candidate. Initially, this holds trivially. So assume that it holds before

a phase. Let c be the east-most candidate of the north-most candidate(s) on the cycle. Since c is either further north than $succ(c)$, or if c is as north as $succ(c)$, it is further east than $succ(c)$, c remains a candidate. Therefore there is no candidate covering c. Hence, c will not withdraw candidacy in subphase 1. In subphase 2, c will not withdraw candidacy because it lies on the outer boundary. Hence, there is still a candidate after the phase. □

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

Proof. Assume that there are two or more candidates in the system. First, we consider the case in which there is a cycle with two or more candidates. Consider any two consecutive candidates on that cycle; if one is further north than the other, at least one of them will withdraw its candidacy in subphase 1. Otherwise, one of them will be further west than the other and therefore will withdraw its candidacy in subphase 1. Hence, the number of candidates will be reduced. Now consider the case that all cycles have at most one candidate. Then there is a cycle corresponding to an inner boundary that has exactly one candidate. That candidate will withdraw candidacy in subphase 2 and thereby reduces the number of candidates in the system. □

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

Theorem 1. *Algorithm 2 deterministically decides the leader election problem in the geometric amoebot model.*

3.5.3 Runtime Analysis

For a cycle of agents let L be the length of the cycle and let l_i be number of leader candidates before phase i of the execution of Algorithm 2. Initially $l_i = L$ since all the agents in the cycle are leader candidates. If $l_i \leq 2$ then in phase $i + 1$ either the leader is elected (outer boundary) or all candidates on the cycle vanish (inner

boundary), as a consequence of Lemma 2. For the case $l_i \geq 2$, Lemma 3 provides the key insight of our analysis.

Lemma 3. *Let l_i be the number of the leader candidates in a cycle. For any phase i, it holds $l_{i+1} \leq \lceil \frac{l_i}{2} \rceil$.*

Proof. Suppose a cycle contains $l_i > 1$ candidates in phase i. For any candidate c in the cycle and its successor candidate, $succ(c)$, the following argument holds. It is clear that either one of these two candidates is further north than the other, or in case one is as north of the other, one is further east than the other. This competition results candidate c or $succ(c)$ to revoke the candidacy. So from every two consecutive candidates in a cycle at least one of them will revoke its candidacy. Thus at the end of subphase 1 of phase i, at least a half of candidates will revoke their candidacy.

□

Theorem 2. *Algorithm 2 terminates in $\mathcal{O}(L_{max} \log(L_{max}))$ time, where L_{max} is the length of the longest cycle in the particle system.*

Proof. Based on previous lemma, Lemma 3, it is easy to see that our algorithm requires $\mathcal{O}(\log(L_{\max}))$ phases to elect a leader. Each phase of the local control algorithm needs $O(l'_i)$ rounds where l'_i is the length of the longest segment. Therefore, Algorithm 2 elects a leader in no worst than $\mathcal{O}(L_{\max} \log(L_{\max}))$ time. □

Improving the Deterministic Leader Election Algorithm Runtime:

Algorithm 2 guarantees that a leader will be elected in $\mathcal{O}(L_{max} \log L_{max})$ time. A small modification of the algorithm can lead to linear runtime, without compromising its correctness. We briefly present a high-level description of this modification: Each candidate c competes with its $succ(c)$ as before, but instead of having only one competition per each phase which is with its first next candidate in the cycle, c continues

the competition with next candidates one after the other in the direction of the cycle, i.e., $succ(succ(c))$ and so on. To do this, the remaining unmatched vectors in c's segment (which made $succ(c)$ to lose the game) continue their way in the direction of the cycle, and try to match with existing tokens in the newly (enlarged) segment of c, which has been enlarged due to revoking candidacy by $succ(c)$. This process continues until either c revokes its candidacy in a competition, or it becomes the only candidate in the cycle (all vectors completely match). It should be noted that the vectors on $seg(c)$ will not pass $succ(c)$ and enter $seg(succ(c))$ except once the candidate $succ(c)$ has lost the game. Considering the east-most candidate out of the north-most candidates, the candidate starts the competition with its first next candidate on the cycle and it beats each candidate on the cycle one after the other. The details for process of withdrawing leadership is as we already mentioned in the randomized version of the algorithm. Very similar idea has been followed in [9] proving our improved runtime to be $\mathcal{O}(L_{\max})$.

Theorem 3. *The modified version of Algorithm 2 as described above requires $O(L_{max})$ rounds to elect a leader.*

We note that we recently got to know that there exists a paper which shares many similarities with our deterministic algorithm. We shortly explain their approach at a high level:
This algorithm works in a synchronous fashion in which cells have access to a "global compass"; i.e., cells have a common sense of where is the "left", "right", "up", and "down" directions. The local control algorithm works on the outer boundary (initially they assume the structure has no hole therefore no inner boundary exists in the structure). Cells only need constant-size memory. In contrast to our algorithm that initially assumes all particles on any boundary are leader candidates, only certain

cells begin as candidates; specifically, starting candidates are those cells that have no particle to their left and above. Using their global compass, each boundary cell can locally determine where it is located relative to the given structure. The cell will accordingly mark itself as "u", "d", "l", and "r" in a local fashion. Figure 3.8, Part (a), shows an initial configuration for the cells, and Part (b), shows those cells with their marks. We omit the detailed explanation of the way each cell determines whether it is a u-cell, or d-cell, etc. These marks are indeed some tokens that each candidate uses to compete. If the number of u-cells, $\mathcal{N}_u$, is more than that of d-cells, $\mathcal{N}_d$, along a the segment of c, c wins the competition. If $\mathcal{N}_u$ and $\mathcal{N}_d$ are equal, the candidate with higher number of r-cells, $\mathcal{N}_r$, will lose the game. These comparisons are made by using a token passing scheme very similar to ours, where each cell tries to forward the token it is holding in the boundary and any two opposite tokens (here "u" versus "d", and "r" versus "l") will be matched once they meet at a cell. Once these comparisons and token deletions are made for all pairs of candidate cells, eventually one candidate will be left and become the leader of the whole structure. The algorithm in [9] works in a synchronous fashion and has linear runtime with respect to the perimeter of the given structure.

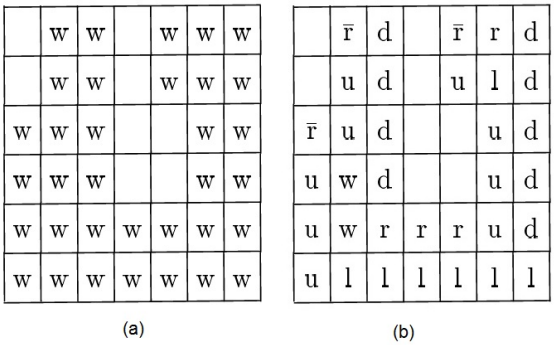

Figure 3.8: Part(a) depicts an example of initial configuration for cells. According to local rules in [9] each cell marks itself as Part(b). Leader candidates are shown by r'.

In Section 4 of this study, the authors outline some required modifications so that their algorithm works for any arbitrary structure, including those that contain holes.

3.6 Conclusions

We considered the geometric variant of the amoebot model by restricting the particle structures to form a connected subset on a triangular grid and showed that for these structures there are local-control protocols for the leader election problem. 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. It is interesting to identify the minimum set of key geometric properties that G must have in order for the proposed algorithms to work.

Chapter 4

SHAPE FORMATION

4.1 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 [50], moteins [21], or nubots [62]. As discussed earlier, our SOPS can be used to envision systems of nano-sensor devices with very limited computational power, but which can collaborate to reach a lot more as a collective. Ideally, those nano-sensor devices will be able to self-organize in order to achieve a desired collective goal without the need of central control or external (in particular, human) intervention. For example, one could envision using a system of self-organizing nano-sensor devices to identify and coat (and possibly repair) leaks on a nuclear reactor without the need for human intervention; self-organizing systems of nano-sensor devices could also be used to monitor environmental and structural conditions in abandoned mines, on the exterior of an airplane or spacecraft, bridges and other structures, possibly also self-repairing the structure— i.e., realizing what has been coined as "smart paint". The applications in the health arena are also endless, e.g., self-organizing nano-sensor devices could be used within our bodies to detect and coat an area where internal bleeding occurs, eliminating the need of immediate surgery, or they could be used to identify and isolate tumor/malignouos cells. In many applications, there may be a specific shape that one would like the system to assume (e.g., a disc, or a line, or even an arbitrary convex shape).

First, we present an initial algorithm for forming a straight line with the set of particles. In Section 4.3, we present a general algorithmic framework for more complex shape formation problems in SOPS, and show direct applications of this framework to the problems of having the particle system self-organize to form a hexagonal or triangular shape. Our algorithms utilize only local control, require only constant-size memory particles, and are asymptotically optimal both in terms of the total number of movements needed to reach the desired shape configuration.

4.1.1 Our Contributions

In this part of the dissertation, we will use the geometric amoebot model presented in Section 2 as our basic model for SOPS. We first present a simple local-control algorithm for the *line formation problem*, where the goal is for any connected structure of particles to eventually form a line. We also present a *general algorithmic framework for shape formation problems* in SOPS, which constitutes of two basic algorithmic primitives: the *spanning forest* primitive and the *snake formation* primitive. We present concrete applications of these two primitives to two specific shape formation problems, namely to the problems of having the system of particles self-organize to form a *hexagonal shape* and to form a *triangular shape*. All the hexagonal shape and the triangular shape and the line formation algorithms are *optimal* with respect to work, which we measure by the total number of particle movements needed to reach the desired shape configuration, as we prove in Theorems 4, 9, and 7. Our algorithms rely only on local information (e.g., particles do not have ids, nor do they know n, the total number of particles, or have any sort of global coordinate/orientation system), and require only constant-size memory particles.

As before, we assume the standard asynchronous model from distributed computing, where the system of particles progresses by performing atomic actions, each of

which affects the configuration of one or two particles. Whenever a particle is activated (i.e., performs an atomic action), 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 at most one movement. A round is over once every particle has been activated at least once.

4.2 Line Formation

In a shape formation problem, the set of final configurations consists of those system configurations 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 configuration. We assume that initially we have an arbitrary connected structure of contracted particles with a unique particle called a *seed*. The seed is used as the starting point for forming the line of particles and specifies the direction in which this line will grow. Particles organize themselves into a spanning set of disjoint trees. 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).

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 seed 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. Recalling from Section 2, 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.

4.2.1 Algorithm

The spanning forest algorithm, given in Algorithm 3, 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 3 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 seed, 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. Figure 4.1 depicts some snapshots of a run of line formation algorithm.

Algorithm 3 Spanning Forest Algorithm for Line Formation Problem

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 $p.linedir$ in the shared memory of edges i and i' and 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

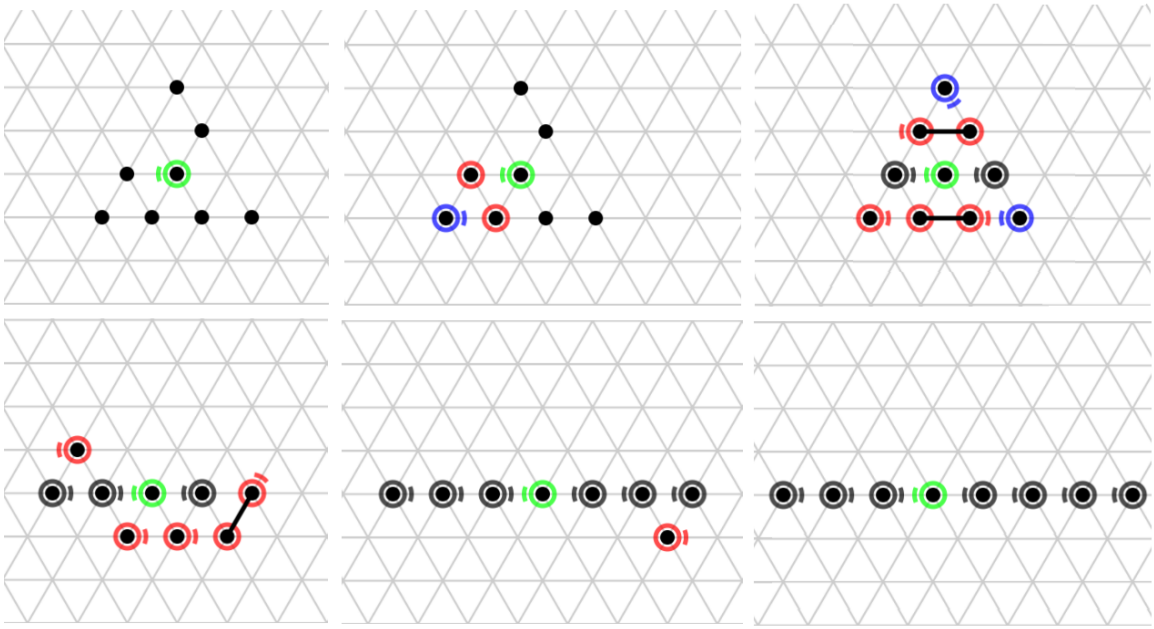

Figure 4.1: Snapshots of the line formation algorithm. The seed particle is green, retired particles are black, roots are red and followers are blue.

In the following section we analyze our line formation algorithm, first analyzing the Spanning Forest algorithm, in general, and then the line formation algorithm.

4.2.2 Analysis for Line Formation Problem

We first generally analyze the correctness of the spanning forest algorithm as the building block we use in all our shape formation problems. We refer to this correctness proof later once it is needed.

Spanning Forest Algorithm Analysis

Once an inactive particle changes its state (into either a root or a follower), we say it becomes *active*. Also, as specified in Algorithm 3, only followers can set the flag $p.parent$. The first three lemmas demonstrate some properties that hold during the execution of the spanning forest procedure and will be used later in Section 4.2.2

to analyze the complete proposed algorithm, when we incorporate the check for retirement of particles according to the line formation problem, and the propagation of the line direction.

Recall that the configuration of the system of particles at time t consists, for every particle p, of the current state of p, including whether the particle is expanded or contracted, of any flags in p's shared memory, and of the node(s) p occupies in G_{eqt} (given by the relative position of p according to the other particles) at time t, as well as the labeling of the bonds of p. Following the asynchronous model, the system of particles progresses by performing atomic actions, each of which affects the configuration of one or two particles.

Lemma 4. *For a follower p, the node indicated by $p.parent$ is occupied by an active particle.*

Proof. Consider a follower p in any configuration during the execution of Algorithm 3. Note that p can only become a follower from an inactive state, and once it leaves the follower state it will not switch to that state again. Consider the first configuration c_1 in which p is a follower. In the configuration c_0 immediately before c_1, p must be inactive and it becomes a follower because of an active particle p' occupying the position indicated by $p.parent$ in c_0. The particle p' is still adjacent to the edge flagged by $p.parent$ in c_1. Now assume that $p.parent$ points to an active particle p' in a configuration c_i, and that p is still a follower in the next configuration c_{i+1} that results from executing an action a. If a affects p and p', the action must be a handover in which p updates its flag $p.parent$ such that $p.parent$ may be moved to the edge that now connects p to p' in c_{i+1}. If a affects p but not p', it must be a contraction in which $p.parent$ does not change and still points to p'. If a affects p' but not p, there are multiple possibilities. The particle p' might switch from follower

to root state, or from root to retired state, or it might expand, none of which violates the lemma. Furthermore, p' might contract. If $p.parent$ points to the head of p', p' is still adjacent to the edge flagged by $p.parent$ in c_{i+1}. Otherwise, p is a child adjacent to the tail of p' in c_i and therefore the contraction must be part of a handover. As p is not involved in the action, the handover must be between p' and a third active particle p''. It is easy to see that after such a handover $p.parent$ points to either p' or p''. Finally, if a affects neither p nor p', $p.parent$ will still point to p' in c_{i+1}. □

Based on Lemma 4, we define a directed graph $A(c)$ for a configuration c as follows. $A(c)$ contains the same nodes as the nodes occupied in G_{eqt} by the set of particles in c. For every expanded particle p in c, $A(c)$ contains a directed edge from the tail to the head of p, and for every follower p' in c, $A(c)$ contains a directed edge from the head of p' to $p'.parent$.

Lemma 5. *The graph $A(c)$ is a forest, and if there is at least one active particle, every connected component of inactive particles contains a particle that is connected to an active particle.*

Proof. In an initial configuration c_0, all particles are inactive and therefore the lemma holds trivially. Now assume that the lemma holds for a configuration c_i. We will show that it also holds for the next configuration c_{i+1} that results from executing an action a. If a affects an inactive particle p, this particle either becomes a follower or a root. In the former case p joins an existing tree, and in the latter case p forms a new tree in $A(c_{i+1})$. In either case, $A(c_{i+1})$ is a forest and the connected component of inactive particles that p belongs to in c_i is either non-existent or connected to p in c_{i+1}. If a affects only a single particle p that is in state follower, this particle can contract or become a root. In the former case, p has no child p' such that $p'.parent$ is the tail of p and also p has no inactive neighbors. Therefore, the contraction of

p does not disconnect any follower or inactive particle and, accordingly, does not violate the lemma. In the latter case, p becomes a root of a tree which also does not violates the lemma. If a involves only a single particle p that is in state root, p can expand or contract or become retired. An expansion and becoming retired trivially cannot violate the lemma and the argument for the contraction is the same as for the contraction of a follower above. Finally, if a involves two active particles in c_i, these particles perform a handover. While such a handover can change the parent relation among the nodes, it cannot violate the lemma. □

The following lemma shows that the spanning forest always makes progress, by showing that as long as the roots keep moving, the remaining particles will eventually follow.

Lemma 6. *An expanded particle eventually contracts.*

Proof. Consider an expanded particle p in a configuration c. Note that p must be active. If there is an enabled action that includes the contraction of p, that action will remain enabled until p eventually contracts when p is validated in the current round. So assume that there is no enabled action that includes the contraction of p. According to Lemma 5 and the transition rule from inactive to active particles, at some point in time all particles in the system will be active. If the contraction of p becomes part of an enabled action before this happens, p will eventually contract. So assume that all particles are active but still p cannot contract. If p has no children, the isolated contraction of p is an enabled action which contradicts our assumption. Therefore, p must have children

Furthermore, p must read at least one child p' having its $p'.parent$ flag pointing towards p over its tail and all children having their parent flags pointing towards p's tail must be expanded as otherwise p could again contract as part of a handover. If p'

would contract, a handover between p' and p would become an enabled action. We can apply the complete argument presented in this proof so far to p' and so on backwards along a branch in a tree in $A(c)$ until we reach a particle that can contract. We will reach such a particle by Lemma 5. Therefore, we found a sequence of expanded particles that starts with p' and ends with a particle that eventually contracts. The contraction of that last particle will allow the particle before it in the sequence to contract and so on. Finally, the contraction of p will become part of an enabled action and therefore p will eventually contract. □

In the above lemmas, the direction of expansion of roots is not used. Furthermore, the fact which particles become roots is also not an issue. Therefore, the algorithm works independently of the selection of roots and their expansion direction. This makes the spanning forest algorithm a reusable algorithmic primitive.

Line Formation Correctness Analysis

Now, we can show that the algorithmic primitives as developed in the Section 4.2 decide the line formation problem.

Theorem 4. *Algorithm 3 correctly decides the line formation problem.*

Proof. We need to show that the algorithm terminates (while each particle decides to halt) and that when it does, the formed shape is a straight line. According to Lemma 5 and due to the transition rule from inactive to active particles, every particle p eventually becomes active, i.e., switches from inactive state. According to spanning forest analysis, Lemmas 4 to 6, as long as the roots keep moving the remaining particles will eventually follow and the spanning forest always makes progress. According to the algorithm proposed for line formation problem, if p is adjacent to the seed particle, it becomes a root and moves in a clockwise manner around the current formed line

structure until it eventually reaches one of the valid positions that can extend the line and becomes retired. By contradiction, assume p never becomes retired. Since the number of particles is bounded (and therefore the size of the current line structure is bounded), there must be an infinite number of configurations c_i where p had a root particle blocking its desired clockwise movement around the line structure. Let p' be the last root p sees as its clockwise neighbor over the line structure (since once a particle becomes a root, it will stay connected to the line structure and always attempt to move in a clockwise manner, p' is well-defined). Applying the same argument inductively to p', we will get an infinite sequence of roots on the line structure that never touch a valid spot pointed by one of $p.linedir$ flags of an already retired particle, a contradiction, since the current line structure (and the number of retired particles) is bounded. Therefore, every root eventually changes into a retired state.

From Algorithm 3, every follower in the neighborhood of a retired particle becomes a root. For every root q with at least one follower child, let c be the first configuration when q becomes retired. If q still has any child in c then all of its children p become roots. Applying this argument recursively we will reach to a configuration such that there exists no root q having a follower child which proves that eventually every follower becomes a root.

Putting it all together, eventually all particles become retired and the algorithm terminates. It should be noted that according to the proposed algorithm once a particle becomes retired it does not move anymore (halts). Note that it also follows from the argument above that the set of retired particles at the end of the algorithm forms a connected structure (since the particles start from a connected configuration and never get disconnected through the process). The connected structure must form a line, since a root particle may only become retired if it is contracted and it now occupies a previously unoccupied spot following the direction of the line. □

Line Formation Work Analysis

Now, we evaluate the performance of our algorithm in terms of the number of movements (expansions and contractions) of the particles, i.e., the total *work* performed by the algorithm.

Lemma 7. *The worst-case work required by Algorithm 3 to solve the line formation problem is* $\Omega(n^2)$.

Proof. Assume the initial configuration of the set of particles forms a connected structure of diameter at most $2\sqrt{n}+2$ (e.g., if it forms a hexagonal or square shape in G_{eqt}). Since the line has diameter equal to $n-1$, there must exist some particle that will need to traverse a distance of at least $\frac{1}{2}n-2\sqrt{n}-3$, a second particle that will traverse a distance of $\frac{1}{2}n-2\sqrt{n}-4$, etc., irrespective of the algorithm used. The number of particle movements incurred will be at least $\sum_{i=1}^{\frac{1}{2}n-2\sqrt{n}-1}(\frac{1}{2}n-2\sqrt{n}-i-2)=\Theta(n^2)$. □

In the following, we will show a matching upper bound:

Theorem 5. *Algorithm 3 terminates in* $O(n^2)$ *work.*

Proof. To prove the upper bound, we simply show that every particle executes $O(n)$ movements. The theorem then follows. Consider a particle p. While p is in inactive or a retired state, it does not move. Let c be the first configuration when p becomes a follower. Consider the directed path in $A(c)$ from the head of p to its root p'. There always is a such a path since every follower belongs to a tree in $A(c)$ by Lemma 5 Let $P=(a_0,a_1,\ldots,a_m)$ be that path in $A(c)$ where a_0 is the head of p and a_m is a child of p'. According to Algorithm 3, p attempts to follow P by sequentially expanding into the nodes $a_0,a_1,\ldots,a_m$. The length of this path is bounded by $2n$ and, therefore, the number of movements p executes while being a follower is $O(n)$. Once p becomes a root, it only performs expansions and contractions around the currently constructed

line structure, until it reaches one of the valid positions on the line. Since the total number of retired particles is at most n, this leads to an additional $O(n)$ movements by p. Therefore, the number of movements a particle p executes is $O(n)$, which concludes the theorem. □

4.3 An Algorithmic Framework for Shape Formation Problems in SOPS

In this section, we focus on solving more complex shape formation problems in the geometric amoebot model starting from any well-initialized connected configuration of particles. We present a general algorithmic framework for shape formation problems and then specifically we investigate the *Hexagonal Shape Formation (HEX)* and the *Triangular Shape Formation (TRI)* problems where the desired shape is a hexagon and a triangle respectively. We formally define a shape formation problem as a tuple $\mathcal{M} = (\mathcal{I}, \mathcal{G})$ where $\mathcal{I}$ and $\mathcal{G}$ are sets of connected configurations of contracted particles. We say $\mathcal{I}$ is the set of permitted initial configurations and $\mathcal{G}$ is the set of goal configurations. We present a general algorithmic framework for shape formation problems and then specifically investigate the *Hexagonal Shape Formation (HEX)* and the *Triangular Shape Formation (TRI)* problems, where $\mathcal{G}$ would be all configurations of contracted particles where the positions of the set of particles induce the largest possible complete hexagon, or the largest possible complete triangle, respectively, on G_{eqt}. We say that an algorithm $\mathcal{A}$ *decides* a shape formation problem $\mathcal{M}$, if for any initial configuration from $\mathcal{I}$, all executions of $\mathcal{A}$ eventually reach one of the valid configurations in $\mathcal{G}$ without losing connectivity, and whenever such a system configuration is reached for the first time, the system stays there and all particles decide to perform no further actions.

As in the line formation problem, for all algorithms we assume that we have a seed particle, which provides the starting point for constructing the respective shape.

We again distinguish for the state of a particle between inactive, follower, root, and retired. Initially, all particles are inactive, except the seed 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. We also assume that every time a particle contracts, it contracts out of its tail.

Generally speaking, the shape formation algorithms we propose progress as follows. Particles organize themselves into a *spanning set of disjoint trees* where the roots of the trees are non-retired particles adjacent to the partially constructed shape structure (consisting of all retired particles). Root particles lead the way by moving in a predefined direction around the current structure.

The remaining particles (i.e., the followers) follow behind the leading root particles, hence the system flattens out towards the direction of movement. Once the leading particles reach a valid position where the shape can be extended (following the rules for the *snake formation* for the particular shape), they stop moving and change their state to retired as well. This process continues until all particles become retired. Note that the spanning forest component of this general approach is the same for any shape formation algorithm: It is only in the rules that determine the next valid position to be filled in the shape structure being built that the respective algorithms differ. We determine the next valid position to be filled sequentially following a *snake* (i.e., a line of consecutive positions in G_{eqt}), that fills in the space of the respective shape structure and scales naturally with the number of particles in the system.

In order to characterize the snake formation for a given shape formation problem, one only needs to specify the direction in which the line of particles forming the snake should continue to grow, for each new particle added to the snake. Hence once a particle finds the next valid position on the snake, it will become retired and set its

snake direction accordingly. Different rules for snake formation will realize different shapes.

4.3.1 Algorithm and Analysis

Spanning Forest Algorithm The Spanning Forest algorithm primitive, given in Algorithm 4, is a building block we use for all of our shape formation problems. This primitive was also used in Section 4.2, where we present a preliminary self-organizing algorithm for forming a straight line of particles. We present the algorithm here for completeness, with the root condition translated to our shape formation framework. Each particle p continuously runs Algorithm 4 until it 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).

Initially all system particles, except the seed, are inactive. In a nutshell, the particles that are touching the seed or other retired particle become roots; the root particles move around the partially constructed shape structure in a clockwise manner until they find a valid position on the snake and become retired; follower particles follow the movement of the respective root until they become roots themselves. As we will see later, the initial snapshots of Figures 4.3 and 4.4 illustrate the spanning forest formation for the respective initial particle configurations.

Note that proofs concerning the general correctness of the spanning forest algorithm have been presented in Section 4.2.2. For space sake, we don't present them again but we refer to them to analyze the complete proposed algorithm.

Algorithm 4 Spanning Forest Algorithm for Shape Formation

Depending on p's current state, a particle p behaves as described below:

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' is expanded, then p expands in the direction given by $p.parent$ in a handover with p', and may need to adjust $p.parent$ to still point to particle 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 runs the corresponding *snake formation algorithm* (Algorithm 5 or 6, for HEX or TRI resp.), and becomes *retired* accordingly. Otherwise, it considers the following three cases: (i) if p is contracted, it tries to expand in the direction given by RootDirection (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.

RootDirection (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

Hexagonal Shape Formation We now investigate the *Hexagonal Shape Formation (HEX)* problem where the system of particles has to assume the shape of a hexagon (but for the outer layer, which may not be completely full) in G_{eqt}. The hexagon will be constructed around the seed particle. Note that a hexagon in G_{eqt} is actually a disk, since it can be defined by the set of all nodes of G_{eqt} within a certain distance r from a seed node.

We will organize the particles according to a spiral snake structure which will incrementally add new layers to the hexagon, scaling naturally with the number of particles in the system. In order to characterize the snake formation for a given shape formation problem, one only needs to specify the direction in which the line of particles forming the snake should continue to grow, for each new particle added to the snake. Hence once a particle finds the next valid position on the snake, it will become retired and set the snake direction accordingly (by correctly setting the flag *p.snakedir* on the respective edge). Different rules for snake formation will realize different shapes. In particular, Algorithm 5 specifies the rules for the spiral snake formation for HEX.

Initially, the seed particle p sets the flag *p.snakedir* in the shared memory corresponding to one of its adjacent edges (e.g., the edge with label 0). Any particle adjacent to a retired particle becomes a root following the spanning forest algorithm. Each root p moves in a clockwise fashion around the structure of retired particles until it finds the next position to extend the hexagonal snake (i.e., a position connected to a retired particle via an edge flagged *p.snakedir*) and becomes retired, following Algorithm 5 (see Figure 4.2).

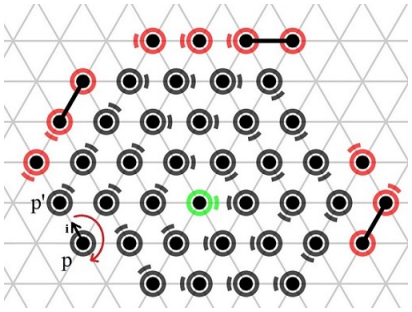

Figure 4.2: shows an intermediate configuration of the HEX algorithm. The seed is depicted in green, retired particles are black, and roots are red. Particle p is the last added particle to the retired structure. Hence, edge i connects p to the retired particle p' (edge i has the flag $p'.snakedir$). The red arrow depicts the process of setting $p.snakedir$ in clockwise manner for p.

Algorithm 5 Snake Formation for HEX

if p is a contracted root **then**

  if p has an adjacent edge i to p' with a flag $p'.snakedir$, where p' is retired **then**

    while edge i is connected to a retired particle **do**

      $i \leftarrow$ label of next edge in clockwise direction

    p sets the flag $p.snakedir$ for edge i

    p becomes *retired*.

Figure 4.3 depicts some snapshots of a run of HEX algorithm.

Analysis for HEX problem

Here, we show that the algorithmic primitives proposed in Section 4.3.1 solve the HEX problem correctly.

Theorem 6. *Our algorithm decides the HEX problem.*

Proof. We need to show that the algorithm terminates and that when it does, the system is in the shape of a hexagon. According to Lemma 5 and the transition rule

from inactive to active particles, every particle p eventually activates. According to spanning forest analysis, Lemmas 4 to 6, as long as the roots keep moving the remaining particles will eventually follow. According to the algorithm proposed for hexagonal shape formation problem, if p is adjacent to the retired structure (initially the structure only contains the seed particle), it becomes a root and moves in a clockwise manner around the retired structure until it eventually reaches the valid position that can extend the hexagon and becomes retired. By contradiction, assume p never becomes retired. Since the number of particles is bounded (and therefore the size of the formed retired structure is bounded), there must be an infinite number of configurations c_i where p had a root particle blocking its desired clockwise movement around the hexagonal retired structure. Let p' be the last root p sees as its clockwise neighbor over the retired structure (since once a particle becomes a root, it will stay connected to the hexagonal retired structure and always attempt to move in a clockwise manner, p' is well-defined). Applying the same argument inductively to p', we will get an infinite sequence of roots on the retired structure that never touch a valid spot pointed by $q.snakedir$ flag of an already retired particle q, a contradiction, since the current retired structure (and the number of retired particles) is bounded. Therefore, every root eventually changes into a retired state. From Algorithm 4, every follower in the neighborhood of a retired particle becomes a root. For every root q with at least one follower child, let c be the first configuration when q becomes retired. If q still has any child in c then all of its children p become roots. Applying this argument recursively we will reach to a configuration such that there exists no root q having a follower child which proves that eventually every follower becomes a root. Putting it all together, eventually all particles become retired and the algorithm terminates and all particles halt.

Note that it also follows from the argument above that the set of retired particles at the end of the algorithm forms a connected structure (since the particles start from a connected configuration and never get disconnected through the process).

Now, we need to prove the correctness, i.e., that the resulting structure of retired particles is in a hexagonal form. Initially the hexagon only contains the seed particle, therefore the claim holds trivially. By induction, let's assume c is the first configuration in which the current formed structure of the retired particles contains k retired particles and by induction hypothesis, assume that those particles form a valid hexagonal shape using k particles. According to Algorithm 5, the only way a root p can become the $(k+1)^{\text{th}}$ retired particle during or after c, is if it occupies the next valid position pointed by the flag $q.snakedir$, where q was the k-th particle to join the hexagonal shape. According to induction hypothesis, the k first retired particles form a hexagonal shape. By pointing to the next adjacent position in counter-clockwise direction around the outermost retired particles in the current hexagonal structure, the flag $q.snakedir$ points to the next position (according to counter-clockwise direction) on the last formed layer of the retired structure, or to a starting position on the next layer once the current layer is full, proving the correctness of the constructed shape. □

We would like to measure the amount of the work of the proposed algorithm.

Lemma 8. *The worst-case work required by any algorithm to solve the HEX problem is* $\Omega(n^2)$.

Proof. Consider a line of n particles on G_{eqt}, where the seed particle is located on one end of the line, as an initial configuration of the particles. We label the particles connected to the seed starting with number 0 for the particle adjacent to the seed. The particle labeled $i > 1$ requires at least $2(i-1-\lceil (i-1)/M_{i-1} \rceil) \geq 2(i-1-\lceil (i-1)/6 \rceil)$

movements until it can lie contracted on the retired structure where M_j, $M_j \geq 6$ and $j \geq 1$, indicates the capacity (i.e., the number of the retired particles) of the layer that the retired particle with label j belongs to. Therefore, any algorithm requires at least $2\sum_{i=2}^{n-1}(i-1-\lceil(i-1)/6\rceil) = \Omega(n^2)$ work. □

Recalling from Section 4.2.2 of the dissertation, and according to Lemma 4, consider the directed graph $A(c)$ for a configuration c as follows. $A(c)$ contains the same nodes as the nodes occupied in G_{eqt} by the set of particles in c. For every expanded particle p in c, $A(c)$ contains a directed edge from the tail to the head of p, and for every follower p' in c, $A(c)$ contains a directed edge from the head of p' to $p'.parent$.

Theorem 7. *The algorithm proposed for HEX terminates in $\mathcal{O}(n^2)$ work.*

Proof. To prove the upper bound, we simply show that every particle executes $\mathcal{O}(n)$ movements. The theorem then follows. Consider a particle p. While p is in inactive or a retired state, it does not move. Let c be the first configuration when p becomes a follower. Consider the directed path in $A(c)$ from the head of p to its root p'. There always is such a path since every follower belongs to a tree in $A(c)$ by Lemma 5 . Let $P = (a_0, a_1, \ldots, a_m)$ be that path in $A(c)$ where a_0 is the head of p and a_m is a child of p'. According to Algorithm 4, p attempts to follow P by sequentially expanding into the nodes $a_0, a_1, \ldots, a_m$. The length of this path is bounded by $2n$ and, therefore, the number of movements p executes while being a follower is $\mathcal{O}(n)$. Once p becomes a root, it only performs expansions and contractions around the retired structure until it reaches one of the valid positions on the hexagon. Since each root p and a retired particle q never connect from the same edge of q more than twice, and since the total number of retired particles is at most n, therefore the number of movements is bound to $\mathcal{O}(n)$ for p. Therefore, the number of movements a particle p totally executes is $\mathcal{O}(n)$, which concludes the theorem. □

Triangular Shape Formation Now we investigate the *Triangular Shape Formation* problem (TRI) where the system of particles has to assume a final triangular shape on G_{eqt} (but for possibly the outer layer).

As we discussed for the HEX problem, in order to solve the TRI problem in our Spanning Forest + Snake Formation algorithmic framework, one only needs to setup the correct rules for growing a "triangular snake", which will be accomplished by Algorithm 6. The snake formation rules for the TRI problem are more complex than the ones we had for the HEX problem, since we will need to explicitly take into account the formation of different layers of particles as we build the triangular structure (this is implicitly taken care by the spiral formation in the HEX algorithm). The TRI snake construction will start from the seed particle p, which will occupy one of the triangle corners. The seed will mark two of its adjacent edges as e_L and e_R, which will determine the direction along which two of the sides (L and R) of the triangle will be formed, by setting $p.border[left]$ and $p.border[right]$ flags on the corresponding edges (we arbitrarily pick the edges will labels 0 and 1 out of p in our algorithm). These directions will be propagated by the particles that end up on L and R resp. The seed starts the snake formation by setting the flag $p.snakedir$ on its 0-labeled edge. From there on, Algorithm 6 will build the triangle snake layer by layer, alternating going "to the left" and "to the right" as the snake places particles on R and L resp. Every time the snake touches one of the sides (Case 2 of Algorithm 6), it sets up the rules for starting a new layer by setting the snake direction flags accordingly, first on the last particle of a layer (the one that just touched the border, in Case 2) and then on the first particle of the newly formed layer (Case 3). If a new layer is not needed, the snake proceeds to fill additional positions on the current layer (Case 1). Figure 4.4 illustrates this approach through some snapshots of the execution of the TRI algorithm.

Algorithm 6 Snake Formation for TRI

if p is a contracted root **then**

if p has an adjacent edge i to p' with a flag $p'.snakedir$, where p' is retired **then** ▷ **retired condition**

$bordertype$ = BORDER (p)

if $bordertype$ = null **then**

▷ **Case 1: continue on the same layer**

p sets $p.snakedir$ for edge opposite to i (i.e., edge $(i+3) \mod 6$)

else

Let q be the border particle connected to p

Let j be the edge of p opposite to the edge connecting p to q

p sets $p.border[bordertype]$ on edge j

if $p' \neq q$ **then**

▷ **Case 2: start a new layer**

p sets $p.snakedir$ for edge j

else

▷ **Case 3: snake direction from border**

if $bordertype = left$ **then**

p sets $p.snakedir$ for edge $(i+5) \mod 6$

else

p sets $p.snakedir$ for edge $(i+1) \mod 6$

p becomes retired

BORDER (p):

if p has an adjacent edge k to a particle q with a flag $q.border[bordertype]$, where $bordertype \in \{left, right\}$ **then**

return $bordertype$

else

return null

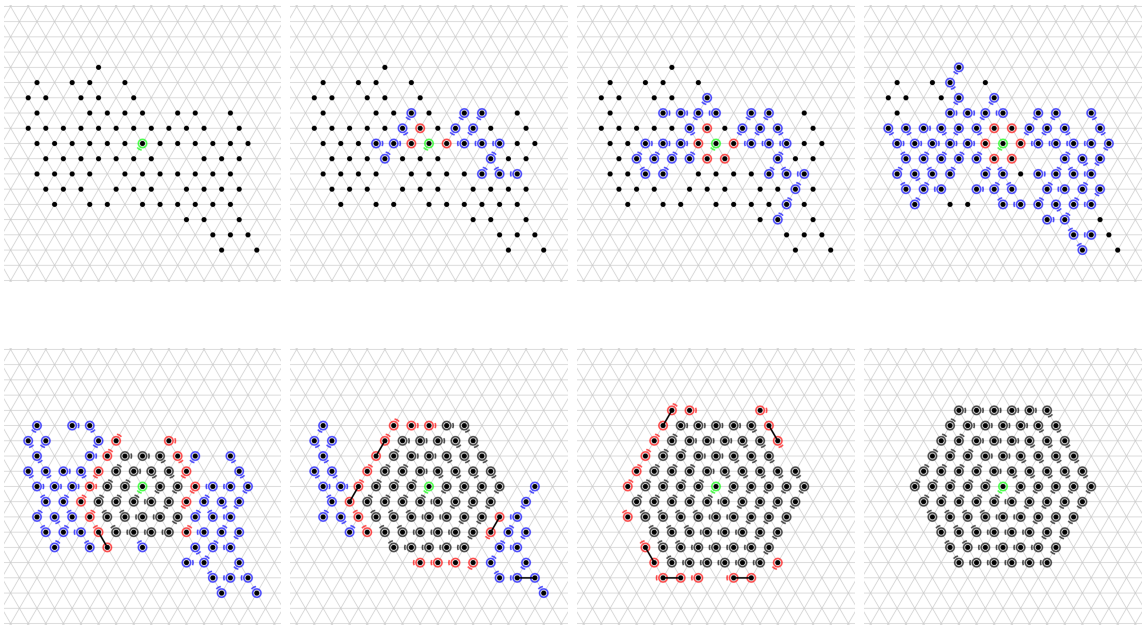

Figure 4.3: Snapshots of the HEX algorithm. The seed is green, retired particles are black, roots are red and followers are blue.

Analysis for TRI problem Now we need to show that the algorithmic primitives presented in Section 4.3.1 decides the TRI problem correctly, in worst-case optimal $O(n^2)$ work.

Theorem 8. *Our algorithm decides the TRI problem.*

Proof. Again, we need to show that the algorithm terminates and that when it does, the system is the shape of a triangle. The termination part of the proof is identical to that for the HEX problem presented in Theorem 6, and hence it only remains to prove the correctness of the TRI algorithm. Assume we have three particles as the base case (to build the smallest size perfect triangle on G_{eqt}). The seed $p*$ sets the $p*.snakedir$ flag and the $p*.border[left]$ flag on its 0-labeled edge. A root particle q might have to move around the seed $p*$ until it connects to edge 0 of the seed through an edges i. Since p sees both (border and snake) flags coming from the same particle, p becomes retired while it start constructing a new layer of the triangle and

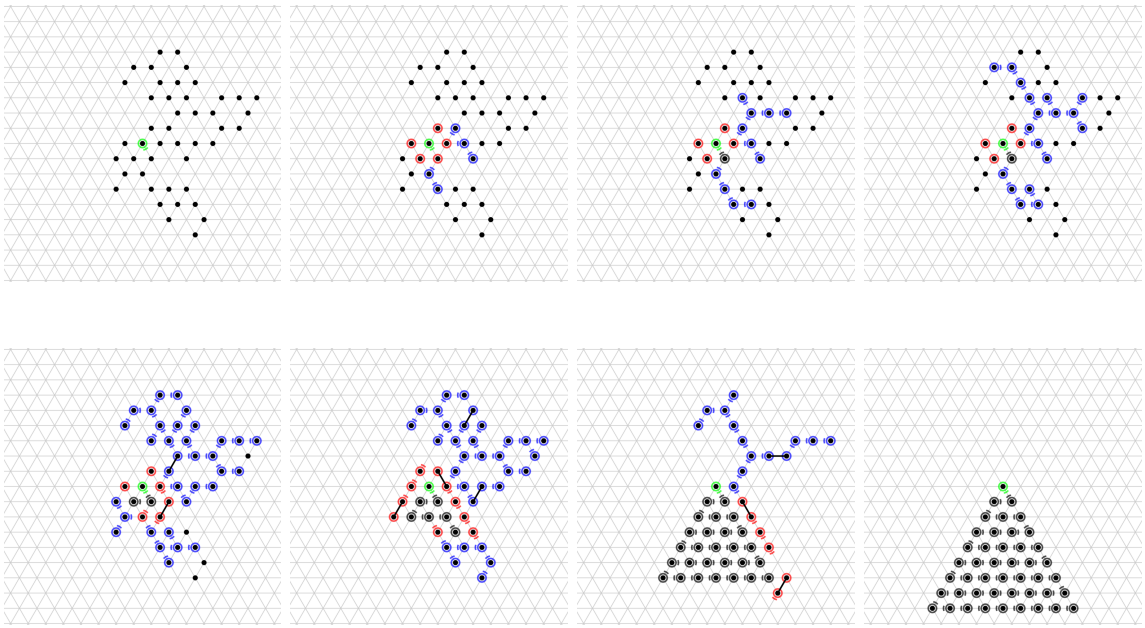

Figure 4.4: Snapshots of the TRI algorithm. The seed is green, retired particles are black, roots are red and followers are blue.

sets its $p.snakedir$ flag such that the next particle continues filling this newly added layer (Case 3 of Algorithm 6). Particle p also sets $p.border[left]$ appropriately to propagate the inherited direction of the border from the seed to next layer. The only position that the third particle can stop on G_{eqt} is the one pointed by $p.snakedir$ and it is trivial to see that the resulting retired structure of the three particles is in a triangular shape. Let c be the first configuration in which the current formed structure of the retired particles contains k retired particles, and let q denote the $(k)^{\text{th}}$ particle to become retired. By induction hypothesis, assume that those k particles form a triangle. According to Algorithm 6, the only way a root p can become the $(k+1)^{\text{th}}$ retired particle during or after c, is if it occupies the valid position pointed by a flag $q.snakedir$. Depending on the location of q in the triangle, three cases may arise. First, consider the case when q is a left border particle (an analogous argument works if q is a right border particle). Since q is the last particle added to the current

valid triangular shape, we either have a perfect triangle after the addition of q or we have a perfect triangle plus particle q as the leftmost particle on a newly created layer. In the former case, given the next position pointed by $q.snakedir$, the root p follows Case 2 of algorithm, which means that p will retire on the leftmost valid position on the next layer of the triangular structure, pointed by $q.border[left]$. In the latter, p follows Case 3 and will fill another position of the current layer next to q. In both cases the resulting retired structure still forms a valid triangular shape. Second, consider the situation where q is not a border particle (Case 1). Therefore, q is located on the last partially filled layer and $q.snakedir$ is set to point to the next unoccupied snake spot on that layer, which is then filled by p, correctly extending the triangular structure, and proving the claim. □

We measure the work of the proposed algorithm as follow:

Lemma 9. *The worst-case work required by any algorithm to solve the TRI problem is* $\Omega(n^2)$.

Proof. With a very similar argument we had in Lemma 8 one can verify that it is required to have at least $2\sum_{i=1}^{n-1}(i-1-\lceil(i-1)/2\rceil) = \Omega(n^2)$ work for the algorithm to terminate. □

Theorem 9. *The algorithm for TRI terminates in* $O(n^2)$ *work.*

Proof. The argument for required amount of work to form a triangular shape will be very similar to that of a hexagonal shape. To prove the upper bound, we show that every particle executes $\mathcal{O}(n)$ movements. The theorem then follows. Consider a particle p. While p is in inactive or a retired state, it does not move. Let c be the first configuration when p becomes a follower. Consider the directed path in $A(c)$ from the head of p to its root p'. There always is such a path since every follower belongs

to a tree in $A(c)$ by Lemma 5 . Let $P = (a_0, a_1, \ldots, a_m)$ be that path in $A(c)$ where a_0 is the head of p and a_m is a child of p'. According to Algorithm 4, p attempts to follow P by sequentially expanding into the nodes $a_0, a_1, \ldots, a_m$. The length of this path is bounded by $2n$ and, therefore, the number of movements p executes while being a follower is $\mathcal{O}(n)$. Once p becomes a root, it only performs expansions and contractions around the retired structure until it reaches one of the valid positions on the triangle. Since each root p and a retired particle q never connect from the same edge of q more than twice, and since the total number of retired particles is at most n, therefore the number of movements is bound to $\mathcal{O}(n)$ for p. Therefore, the number of movements a particle p totally executes is $\mathcal{O}(n)$, which concludes the theorem. □

4.4 Conclusions

We presented a simple algorithm for forming a straight line of particles in geometric amoebot model. We analyzed the correctness of the algorithm and evaluated its performance with respect to worst-case number of particles movements required for termination. We then presented a general algorithmic framework for more complex shape formation problems in SOPS that combines our spanning forest algorithmic primitive with a snake formation primitive. We have shown that by carefully determining how to grow the appropriate snake structure, we were able to solve the HEX and TRI problems. We can easily extend our snake primitive to build other shapes, such as square or rectangular shapes. It would be interesting to characterize all the general shapes that could be solved with our approach on G_{eqt} (and possibly also for other infinite regular grid graphs, namely the square grid graph and the hexagonal grid graph, if we considered those as the underlying graph G in the geometric amoebot model). Finally, we evaluated the performance of our HEX and TRI algorithms in terms of the worst-case amount of work necessary for termination.

In [33] we study another interesting problem related to shape formation in SOPS where we present a *universal* shape formation algorithm which takes an arbitrary shape composed of a constant number of equilateral triangles of unit size and lets the particles build that shape at a scale depending on the number of particles in the system.

Chapter 5

IMPOSSIBILITY RESULTS IN GENERAL AMOEBOT MODEL

5.1 Introduction

As we have seen in Section 4, a central problem for programmable matter is shape formation. In order to determine how decentralized shape formation can be handled, we are particularly interested in the connection between leader election and shape formation. 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 section 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 using a local-control protocol that we studied in Section 4.2. Here we discuss the limitations of solving these problems within the general amoebot model. Of course, in the general amoebot model a straight line is not well-defined. Hence, for this model the set of final configurations for the line formation problem is defined to consist of all system configurations in which the particles form a simple path in G. Note that the leader election problem is well defined for the general amoebot model.

5.2 Impossibility Results

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 configuration). Since in this case it is possible to decide when $G|_A$, where $A \subseteq V$ is the set of occupied nodes in G_{eqt}, 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 [42] 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 [42] 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 [42] 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, as we showed in Section 3, for the geometric amoebot model 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. We can use this algorithm to choose a unique leader as the seed particle for the line formation problem.

5.3 Conclusions

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 have shown 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.

Chapter 6

UNIVERSAL COATING

6.1 Introduction

Today, engineers often need to visually inspect bridges, tunnels, wind turbines and other large civil engineering structures for defects — a task that is both time-consuming and costly. In the not so distant future, *smart coating* technology could do the job faster and cheaper, and increase safety at the same time. The idea behind smart coating (also coined *smart paint*) is to have a thin layer of a specific substance covering the object so that one can measure a certain condition (like temperature or cracks) at any spot on the surface of the object without requiring direct access to that spot. Also in nature, smart coating occurs in various situations. Prominent examples are proteins closing wounds, antibodies surrounding bacteria, or ants surrounding food in order to transport it to their nest. So one can envision a broad range of coating applications for programmable matter in the future. We intend to study coating problems in the context of self-organizing programmable matter consisting of simple computational elements, called particles, that can establish and release bonds and can actively move in a self-organized way. As a basic model for these self-organizing particle systems, we will use the geometric version of the amoebot model presented in Section 2.

6.1.1 Universal Coating Problem

In the *universal coating problem* we consider an instance (P, O) where P represents the particle system and O represents the fixed object to be coated. Let $V(P)$ be the

set of nodes occupied by P and $V(O)$ be the set of nodes occupied by O (when clear from the context, we may omit the $V(\cdot)$ notation). Then let the set of nodes in G_{eqt} neighboring O be called the *surface (coating) layer*. Let n be the number of particles and B_1 be the number of nodes in the surface layer. For any two nodes $v, w \in V_{\text{eqt}}$, the *distance* $d(v, w)$ between v and w is the length of the shortest path in G_{eqt} from v to w. The distance $d(v, U)$ between a $v \in V_{\text{eqt}}$ and $U \subseteq V_{\text{eqt}}$ is defined as $\min_{w \in U} d(v, w)$. An instance is *valid* if the following properties hold:

1. The particles are all contracted and are initially in an *idle* state.

2. The subgraphs of G_{eqt} induced by $V(O)$ and $V(P) \cup V(O)$, respectively, are connected, i.e., there is a single object and the particle system is connected to the object.

3. The subgraph of G_{eqt} induced by $V_{\text{eqt}} \setminus V(O)$ is connected, i.e., the object O has no holes.[1]

4. $V_{\text{eqt}} \setminus V(O)$ is $2(\lceil \frac{n}{B_1} \rceil + 1)$-connected, i.e. O cannot form *tunnels* of width less than $2(\lceil \frac{n}{B_1} \rceil + 1)$.

Note that a width of at least $2\lceil \frac{n}{B_1} \rceil$ is needed to guarantee that the object can be evenly coated. See Figure 6.1 for an example of an object with a tunnel of width 1. The coating of narrow tunnels requires specific technical mechanisms that complicate the protocol without contributing to the basic idea of coating, so we ignore such cases in favor of simplicity and a clean presentation.

A configuration C is *legal* if and only if all particles are contracted and

$$\min_{v \in V_{\text{eqt}} \setminus (V(P) \cup V(O))} d(v, V(O)) \geq \max_{v \in V(P)} d(v, V(O))$$

[1] If O does contain holes, we consider the subset of particles in each connected region of $V_{\text{eqt}} \setminus V(O)$ separately.

Figure 6.1: An example of an object with a tunnel of width 1.

meaning that all particles are as close to the object as possible or *coat O as evenly as possible*. If the object has to be coated by more than one layer of particles then the *i-th layer* around the object are the nodes that have a distance of i to the object.

An algorithm *solves* the universal coating problem if, starting from any valid configuration, it reaches a *stable legal configuration C* in a finite number of rounds. A configuration C is said to be stable if no particle in C ever performs a state change or movement.

6.1.2 Our Contributions

Our main contribution in this section is a *worst-case work-optimal* algorithm for the universal coating problem on self-organizing particle systems. Our *Universal Coating Algorithm* seamlessly adapts to any valid object O, uniformly coating the object by forming multiple coating layers (where each coating layer consists of equidistant particles to the object) if necessary. As stated in Section 2, our particles are anonymous, do not have any global information (including on the number of particles n), have constant-size memory, and utilize only local interactions.

Our algorithm builds upon many primitives, some of which may be of interest on their own: The *spanning forest* primitive organizes the particles into a spanning forest which is used to guide the movement of particles while preserving connectivity in the system; the *complaint-based coating* primitive allows the first layer to form,

only expanding the coating of the first layer as long as there is still room and there are particles still not touching the object; the *general layering* primitive allows the layer ℓ to form only after layer $\ell - 1$ has been completed, for $\ell \geq 2$; and a *node-based leader election* primitive that works even as particles move and that is used to jumpstart the general layering process. One of the main contributions of our work is to show how these asynchronous primitives can be integrated in a seamless way, with no underlying synchronization mechanisms.

Section 6.2 describes our Universal Coating algorithm. A formal correctness and a worst-case work analyses of the algorithm follow in Section 6.3. We present our concluding remarks in Section 6.5.

6.2 Universal Coating Algorithm

In this section we present our Universal Coating algorithm: In Section 6.2.1, we introduce some preliminary notions; Section 6.2.2 introduces the algorithmic primitives used for the coating algorithm; and lastly Section 6.2.3 focuses on the leader election process needed in certain instances of the problem.

6.2.1 Preliminaries

We define the set of *states* that a particle can be in as *idle*, *follower*, *root*, and *retired*. In addition to its state, a particle maintains a constant number of other flags, which in our context are constant size pieces of information visible to neighboring particles. A flag x owned by some particle p is denoted by $p.x$. While particles are anonymous, when a particle p sets a flag of type x in its shared memory, we will denote it by $p.x$ (e.g., $p.parent$, $p.dir$, etc.), so that ownership of the respective flag becomes clear. In our proposed algorithm, we assume that every time a particle contracts, it contracts out of its tail.

Therefore, a node occupied by the head of a particle p still is occupied by p after a contraction.

We define a *layer* as the set of nodes v in G_{eqt} that are equidistant to the object O. More specifically a node v is in layer ℓ if $d(v, V(O)) = \ell$; in particular the surface coating layer defined earlier corresponds to layer 1. A particle keeps track of its current layer number in $p.layer$. In order to respect the constant-size memory constraint of particles, we take all layer numbers modulo 4. Any root or retired particle p stores a flag $p.layer$ indicating the layer number of the node occupied by the head of p. We say that layer i, $i \geq 1$, is *complete* if each node in that layer is occupied with a retired particle (except for the last layer which can be partially filled with retired particles). However, for ease of presentation, we will omit the modulo 4 computations in the text, except for the pseudocode description of the algorithms.

Each root particle p has a flag storing a bond label $p.down$ pointing to an occupied node adjacent to its head in layer $p.layer-1$ or in the object if $p.layer = 1$. Moreover, p has two additional flags, $p.CW$ and $p.CCW$, which are also bond labels. Intuitively, if p continuously moves by expanding in direction $p.CW$ (resp., $p.CCW$) and then contracting, it moves along a *clockwise* (resp. *counter-clockwise*) path around the connected structure consisting of the object and retired particles. Formally, $p.CW$ is the label of the first port to a node v in *counter-clockwise (CCW) order* from $p.down$ such that either v is occupied by a particle q with $q.layer = p.layer$, or v is unoccupied (in the latter, v may be a node on layer $p.layer$ or $p.layer - 1$). We define $p.CCW$ analogously, following a *clockwise (CW) order* from $p.down$. Figure 6.2 illustrates the different layers around an object, and also CW and CCW traversals of those.

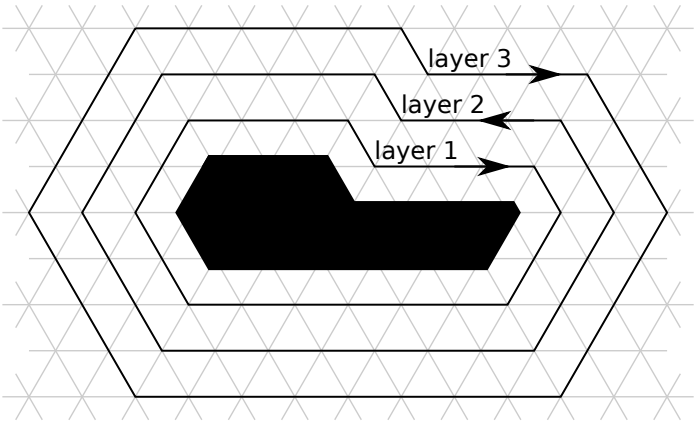

Figure 6.2: We illustrate the first three coating layers with respect to the given object (represented by the nodes in G_{eqt} shaded in black); we also illustrate the direction in which these layers will be filled by our algorithm – CW for odd layers, and CCW for even layers – as determined in Section 6.2.2.

6.2.2 Coating Primitives

Our algorithm can be decomposed into a set of primitives, which are all concurrently executed by the particles, as we briefly explained in Section 7.1.1. Namely the algorithm relies on the following key primitives: the *spanning forest primitive*, the *complaint-based coating primitive* used to establish the first layer of coating, the *general layering primitive*, and a *node-based* (rather than particle-based) *leader election* primitive that works even as particles move, and that is used to jumpstart the general layering primitive. One of the main contributions of our work is to show how these primitives can be put to work together in a seamless way and with no underlying synchronization mechanisms.[2]

The ***spanning forest primitive*** (Algorithm 7) organizes the particles in a spanning forest, in which the roots of the trees will be in state *root* and will determine

[2]A video illustrating a fully asynchronous execution of our universal coating algorithm can be found in [1].

the direction of movement which is specified by a port label $p.dir$; the remaining non-retired particles follow the root particles using handovers. The main benefit of organizing the particles in a spanning forest connected to the surface is that it provides a relatively easy mechanism for particles to move, following the tree paths, while maintaining connectivity in the system (see [36, 32] for more details). All particles are initially *idle*. A particle p becomes a *follower* when it sets a flag $p.parent$ corresponding to the port leading to its parent in the spanning forest (any adjacent particle q to p can then easily check if q is a child of p). As the root particles find final positions according to the partial coating of the object, they stop moving and become retired. Namely, a root particle p becomes *retired* when it encounters another retired particle across the direction $p.dir$.

Algorithm 7 Spanning Forest Primitive

A particle p acts depending on its state as described below:

idle: If p is connected to the object O, it becomes a *root* particle, makes the current node it occupies a *leader candidate position*, and starts running the leader election algorithm described in Section 6.2.3. If p is connected to a *retired* particle, p also becomes a *root* particle. If an adjacent particle p' is a root or a follower, p sets the flag $p.parent$ to the label of the port to p', puts a *complaint flag* in its local memory, and becomes a *follower*. If none of the above applies, p remains idle.

follower: If p is contracted and connected to a retired particle or to O, then p becomes a *root* particle. If p is contracted and has an expanded parent, then p initiates HANDOVER(p); Otherwise, if p is expanded, it considers the following two cases: (i) if p has a contracted child particle q, then p initiates HANDOVER(p); (ii) if p has no children and no idle neighbor, then p contracts. Finally, if p is contracted, it runs the function FORWARDCOMPLAINT($p, p.parent$) described in Algorithm 9.

root: If particle p is on the surface coating layer, p participates in the leader election process described in Section 6.2.3. If p is contracted, it first executes MARKERRETIREDCONDITIONS(p) (Algorithm 12), and becomes *retired*, and possibly also a *marker*, accordingly; if p does not become retired, if it has an expanded root in $p.dir$, then p initiates HANDOVER(p). Otherwise p calls LAYEREXTENSION (p) (Algorithm 10). If p is expanded, it considers the following two cases: (i) if p has a contracted child, then p initiates HANDOVER(p); (ii) if p has no children and no idle neighbor, then p contracts. Finally, if p is contracted, it runs FORWARDCOMPLAINT($p, p.dir$) (Algorithm 9).

retired: p clears a potential complaint flag from its memory and performs no further action.

Recall that B denotes the number of nodes on the surface coating layer (layer 1). We need to ensure that once $\min\{n, B\}$ particles are on layer 1, they stop moving and the coating is complete, independent of how B compares to n (i.e., whether $n \leq B$ or not). In order to be able to seamlessly adapt to all possible surface configurations, we use our novel ***complaint-based coating primitive*** for the first layer, which basically translates into having the root particles (touching the object) open up one more position on layer 1 only if there exists a follower particle that remains in the system. This is accomplished by having each particle that becomes a follower generate a *complaint flag*, which will be forwarded by particles in a pipeline fashion from children to parents through the spanning forest and then from a root q to another root at $q.dir$, until it arrives at a root particle p with an unoccupied neighboring node at $p.dir$ (we call such a particle p a *super-root*). Upon receiving a complaint flag, a super-root p consumes the flag and expands into the unoccupied node at $p.dir$. The expansion will eventually be followed by a contraction of p, which will induce a series of expansions and contractions of the particles on the path from p to a follower particle z, eventually freeing a position on the surface coating layer to be taken by z. In order to ensure that the consumption of a complaint flag will indeed result in one more follower touching the object, one must give higher priority to a follower child particle in a handover operation, as we do in Algorithm 8. The complaint-based coating phase of the algorithm will terminate either once all complaint flags are consumed or when layer 1 is filled with contracted particles. In either case, the particles on layer 1 will move no further. Figure 6.3 illustrates the complaint-based coating primitive.

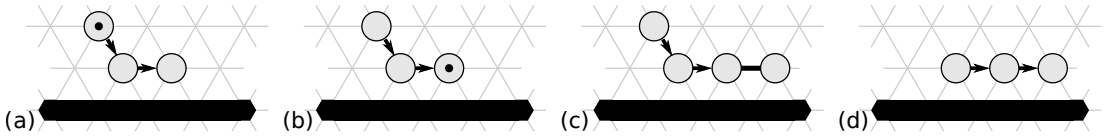

Figure 6.3: Complaint-based coating primitive: Particles are shown as grey circles. In (a), a follower particle generates a complaint flag (depicted as a black dot within the particle) that is then forwarded to a super-root (b) causing the super-root to expand into an unoccupied node (c). After a series of handovers, the follower particle that generated the complaint flag can move to a position on the surface (d).

Algorithm 8 HANDOVER (p)

1: **if** p is expanded **then**
2: **if** $p.layer = 1$ and p has a follower child q such that $q.parent$ points to the tail of p **then**
3: **if** q is contracted **then**
4: p initiates a handover with particle q
5: **else**
6: **if** p has any contracted (follower or root) child q such that $q.parent$ points to the tail of p **then**
7: p initiates a handover with particle q
8: **else**
9: **if** p has an expanded parent q or the position in $p.dir$ is occupied by an expanded root q **then**
10: p initiates a handover with particle q

Algorithm 9 ForwardComplaint(p, i)

1: **if** p holds at least a complaint flag **and** the particle adjacent to p in direction i holds less than two complaint flags **then**

2: p forwards the complaint flag to the particle given by i

Once layer 1 is complete and if there are still follower particles in the system, the ***general layering primitive*** steps in, which will build further coating layers. We accomplish this by electing a *leader marker particle* on layer 1 (via the ***leader election primitive*** proposed in Section 6.2.3). This leader marker particle will be used to determine a "beginning" (and an "end") for layer 1 and allow the particles on that layer to start retiring according to the retired condition given in Algorithm 12 (the leader marker particle will be the first retired particle in the system). Once a layer ℓ becomes completely filled with retired (contracted) particles, a new marker particle will emerge on layer $\ell+1$, and start the process of building this layer (i.e., start the process of retiring particles on that layer) according to Algorithm 12. A marker particle on layer $\ell+1$ only emerges if a root particle p connects to the marker particle q on layer ℓ via its marker port and if q verified locally that layer ℓ is completely filled (by checking whether $q.CW$ and $q.CCW$ are both retired).

With the help of the marker particles — which can only be established after layer 1 was completely filled (and hence, we must have $B \leq n$) — we can replace the complaint-based coating algorithm of layer 1 with a simpler coating algorithm for the higher layers, where each root particle p just moves in CW or CCW direction (depending on its layer number) until p encounters a retired particle on the respective layer and retires itself. More precisely, each contracted root particle p on layer ℓ tries to extend this layer by expanding into an unoccupied position on layer ℓ, or by moving into an unoccupied position in layer $\ell - 1$ (when $p.layer$ will change to $\ell - 1$

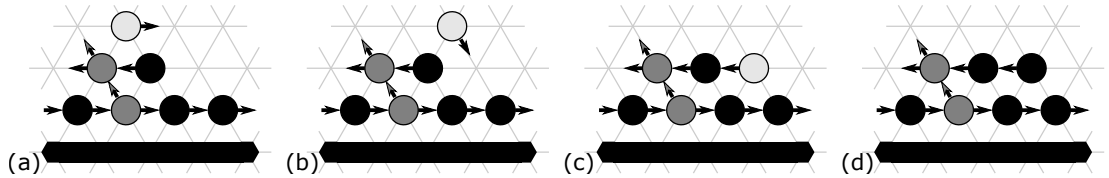

Figure 6.4: General layering primitive: Retired particles are shown as black circles, other than (retired) marker particles which are shown in dark grey (the dark grey arrows represent the marker edges); a root particle is depicted in light grey. Black arrows show the current direction of movement (given by the *dir* flag) for each particle (which becomes irrelevant once a particle retires). (a) The root particle p is located on layer $\ell = 3$; (b) particle p moves in CW direction over retired particles on layer $\ell - 1$; (c) after a series of expansions and contractions following $p.dir$, p arrives at an unoccupied neighboring node on layer $\ell - 1$; (d) since $p.dir$ leads to a retired particle, p retires too.

accordingly), following the direction of movement given by $p.dir$. Figure 6.4 illustrates this process. The direction $p.dir$ is set to $p.CW$ (resp., $p.CCW$) when $p.layer$ is odd (resp., even). Alternating between CCW and CW movements for the particles in consecutive layers ensures that a layer ℓ is completely filled with retired particles before particles start retiring in layer $\ell + 1$, which is crucial for the correctness of our layering algorithm.

Algorithm 10 LayerExtension (p)

Calculating $p.layer$**,** $p.down$ **and** $p.dir$

The layer number of any node occupied by the object is equal to 0.

Let q be any neighbor of p with smallest layer number (modulo 4).

$p.down \leftarrow$ p's label for port leading to q

$p.layer = (q.layer + 1) \mod 4$

clockwise $(p, p.down)$ ▷ Computes CW & CCW directions

if $p.layer$ is *odd* **then**

    $p.dir \leftarrow p.CW$

else

    $p.dir \leftarrow p.CCW$

Extending layer $p.layer$

if the position at $p.dir$ is unoccupied, and either p is not on the first layer, or p holds a complaint flag **then**

    p expands in direction $p.dir$

    p consumes its complaint flag, if it holds one

Algorithm 11 CLOCKWISE (p, i)

1: $j \leftarrow i, k \leftarrow i$

2: **while** edge j is connected to the object or to a retired particle with layer number $p.layer - 1$ **do**

3: $\quad j \leftarrow (j-1) \mod 6$

4: $p.CW \leftarrow j$

5: **while** edge k is connected to the object or to a retired particle with layer number $p.layer - 1$ **do**

6: $\quad k \leftarrow (k+1) \mod 6$

7: $p.CCW \leftarrow k$

Algorithm 12 MarkerRetiredConditions(p)

First marker condition:

if p is *leader particle* **then**

    p becomes a *retired* particle

    p sets the flag $p.marker$ to be the label of a port leading to a node guaranteed not to be on layer $p.layer$ — e.g., by taking the average direction of p's two neighbors in layer 1 (by now complete)

Extending Layer Markers:

if p is connected to a marker q and the port $q.marker$ points towards p **then**

    if both $q.CW$ and $q.CCW$ are retired **then**

        p becomes a *retired* particle

        p sets the flag $p.marker$ to the label of the port opposite of the port connecting p to q

Retired Condition:

if edge $p.dir$ is occupied by a retired particle **then**

    p becomes *retired*

6.2.3 Leader Election Primitive

In this section, we describe the process used for electing a leader among the particles that touch the object. Note that only particles in layer 1 will ever participate in the leader election process. A leader will only emerge if $B \leq n$; otherwise the process will stop at some point without a leader being elected. As discussed earlier, a leader is elected on layer 1 to provide a "checkpoint" (a *marker* particle) that the particles can use in order to determine whether the layer has been completely filled (and a leader is only elected after this happens).

The leader election algorithm we use here is similar to the algorithm presented in Section 3 (and also in [26]) with the difference that leader candidates are associated with positions instead of particles (which is important because in our case particles may still move during the leader election primitive) as we presented in [34]. Hence, for the purpose of universal coating, we will abstract the leader election algorithm to conceptually run on the *nodes* in layer 1, and not on the particular particles that may occupy these nodes at different points in time. The particles on layer 1 will simply provide the means for running the leader election process on the respective positions, storing and transferring all the flags (which can be used to implement the tokens described in [36, 26]) that are needed for the leader competition and verification. An expanded particle p on layer 1, whose tail occupies node v in layer 1, that is about to perform a handover with contracted particle q will pass all the information associated with v to q using the particles' local shared memories. If a particle p occupying position v would like to forward some leader election information to a node w adjacent to v that is currently unoccupied, it will wait until either p itself expands into w, or another particle occupies node w. It is important to note that according to the complaint-based coating algorithm that we run on layer 1, if a node

v in layer 1 is occupied at some time t, then v will never be left unoccupied after time t. Here we outline the differences between the leader election process used in this section and that of Section 3).

- Only the nodes on layer 1 that initially hold particles start as *leader node candidates*. Other nodes in layer 1 will take part in the leader node election process by forwarding any tokens between two consecutive leader node candidates, as determined for the leader election process for a set of static particles forming a cycle in [36]. Note that layer 1 is a cycle on G_{eqt}.
- The leader election process will determine which leader node candidate in layer 1 will emerge as the unique *leader node*. The *leader particle* is then chosen as described below.
- If particle p is expanded, it will hold the flags and any other information necessary for the leader election process corresponding to each node p occupies (head and tail nodes) independently. In other words, an expanded particle emulates the leader election process for two nodes on the surface layer.
- A particle p occupying node v forwards a flag τ to the node w in CW (or CCW) direction along the surface layer only if node w is occupied by a particle q (note that q may be equal to p, if p is expanded) and q has enough space in its (constant-size) memory associated with node w; otherwise p continues to hold the flag τ in its shared memory associated with node v.
- If p is expanded along an edge (v, w) and wants to contract into node w, there must exist a particle q expanding into v (due to the complaint-based mechanism), and hence p will transfer all of its flags currently associated with node v to q.

After the solitude verification phase in the leader election algorithm of [26, 36] is complete, there will be just one leader node v left in the system. Once v is elected a

leader node and there are no more followers, or all positions in layer 1 are filled by contracted particles, then whatever particle currently covers that position becomes the *leader*. In order to check if layer 1 is completely filled with contracted particles, a contracted particle p occupying this position will follow the following process: when a contracted particle p occupies node v it will generate a single *CHK* flag which it will forward to its CCW neighbor q *only if* q *is contracted*. Any particle q receiving a *CHK* flag will also only forward the flag to its CCW neighbor z if and only if z is contracted. If the *CHK* flag at a particle q ever encounters an expanded CCW neighbor, the flag is held back until the neighbor contracts. Additionally, the particle at position v sends out a *CLR* flag to its CW neighbor as soon as it expands. This flag is always forwarded in CW direction. If a *CLR* and a *CHK* flag meet at some particle, the flags cancel each other out. If at some point in time, a particle p at node v receives a *CHK* flag from its CW neighbor in layer 1, it implies that layer 1 must be completely filled with contracted particles (and the complaint-based algorithm for layer 1 has converged), and at that time this contracted particle p elects itself the *leader particle*, setting the flag $p.leader$. Note that the leader election process itself does not incur any additional particle expansions or contractions on layer 1, only the complaint-based algorithm does. If the primitive does not terminate (which only happens if $n < B_1$ and layer 1 is never completely covered), then the complaint flags ensure that the super-roots eventually stop, which eventually results in a stable legal coating.

6.3 Correctness and Work Analysis

In this section we show that our algorithm eventually solves the coating problem, and we bound its worst-case work.

We say a particle p' is the *parent* of particle p if p' occupies the node in direction $p.parent$. Let an *active* particle be a particle in either follower or root state. We call an active particle a *boundary particle* if it has the object or at least one retired particle in its neighborhood, otherwise it is a *non-boundary particle*. A boundary particle is either a root or a follower, whereas non-boundary particles are always followers. Note that throughout the analysis we ignore the modulo computation of layers done by the particles, i.e., layer 1 is the unique layer of nodes with distance 1 to the object.

Given a configuration C, we define a directed graph $A(C)$ over all nodes in G_{eqt} occupied by active particles in C. For every expanded active particle in C, $A(C)$ contains a directed edge from the tail to the head node of p. For every non-boundary particle p, $A(C)$ has a directed edge from the head of p to $p.parent$, if $p.parent$ is occupied by an active particle, and for every boundary particle p, p has a directed edge from its head to the node in the direction of $p.dir$ as it would be calculated by Algorithm 10, if $p.dir$ is occupied by an active particle. The *ancestors* of a particle p are all nodes reachable by a path from the head of p in $A(C)$. For each particle p we denote the ancestor that has no outgoing edge with $p.superRoot$, if it exists. Certainly, since every node has at most one outgoing edge in $A(C)$, the nodes of $A(C)$ can only form a collection of disjoint trees or a ring of trees. We define a *ring of trees* to be a connected graph consisting of a single directed cycle with trees rooted at it.

First, we prove several safety conditions, and then we prove various liveness conditions that together will allow us to prove that our algorithm solves the coating problem.

6.3.1 Safety

Suppose that we start with a valid instance (P, O), i.e., all particles in P are initially contracted and idle and $V(P) \cup V(O)$ forms a single connected component in G_{eqt}, among other properties. Then the following properties hold, leading to the fact that $V(P) \cup V(O)$ stays connected at any time.

Lemma 10. *At any time, the set of retired particles forms completely filled layers except for possibly the current topmost layer ℓ, which is consecutively filled with retired particles in CCW direction (resp. CW direction) if ℓ is odd (resp. even).*

Proof. From our algorithm it follows that the first particle that retires is the leader particle, setting its marker flag in a direction not adjacent to a position in layer 1. The particles in layer 1 then retire starting from the leader in CCW direction around the object. Once all particles in layer 1 are retired, the first particle to occupy the adjacent position to the leader via its marker flag direction will retire and become a marker particle on layer 2, extending its marker flag in the same direction as the flag set by the marker (leader) on layer 1. Starting from the marker particle in layer 2, other contracted boundary particles can retire in CW direction along layer 2. Once all particles in layer 2 are retired, the next layer will start forming. This process continues inductively, proving the lemma. □

The next lemma characterizes the structure of $A(C)$.

Lemma 11. *At any time, $A(C)$ is a forest or a ring of trees. Any node that is a super-root (i.e., the root of a tree in $A(C)$) or part of the cycle in the ring of trees is connected to the object or to a retired particle.*

Proof. An active particle can either be a follower or a root. First, we show the following claim.

Claim 1. *At any time, $A(C)$ restricted to non-boundary particles forms a forest.*

Proof. Let $A'(C)$ be the induced subgraph of $A(C)$ by the non-boundary particles only. Certainly, at the very beginning, when all particles are still idle, the claim is true. So suppose that the claim holds up to time t. We will show that it then also holds at time $t+1$. Suppose that at time $t+1$ an idle particle p becomes active. If it is a non-boundary particle (i.e., a follower), it sets *p.parent* to a node occupied by a particle q that is already active, so it extends the tree of q by a new leaf, thereby maintaining a tree. Edges can only change if followers move. However, followers only move by a handover or a contraction, thus a handover can only cause a follower and its incoming edges to disappear from $A'(C)$ (if that follower becomes a boundary particle), and an isolated contraction, can only cause a leaf and its outgoing edge to disappear from $A'(C)$, so a tree is maintained in $A'(C)$ in each of these cases. □

Next we consider $A(C)$ restricted to boundary particles.

Claim 2. *At any time, $A(C)$ restricted to boundary particles forms a forest or a ring.*

Proof. The boundary particles always occupy the nodes adjacent to retired particles or the object. Therefore, due to Lemma 10, the boundary particles either all lie in a single layer or in two consecutive layers. Since the layer numbers uniquely specify the movement direction of the particles, connected boundary particles within a layer are all connected in the same orientation. Therefore, if these particles all lie in a single layer, they can only form a directed list or directed cycle in $A(C)$, proving the claim. If they lie in two consecutive layers, say, ℓ and $\ell-1$, then $\ell-1$ must contain at least one retired particle, so the nodes occupied by the boundary particles in layer $\ell-1$ can only form a directed list. If there are at least two boundary particles in layer $\ell-1$, this must also be true for the nodes occupied by the boundary particles in layer ℓ because according to Lemma 10 there must be at least two consecutive nodes in

layer $\ell - 1$ not occupied by retired particles. Moreover, it follows from the algorithm that $p.dir$ of a boundary particle can only point to the same or the next lower layer of p, implying that in this case $A(C)$ restricted to the nodes occupied by all boundary particles forms a forest. □

Since a boundary particle p never connects to a non-boundary particle the way $p.dir$ is defined, and a follower without an outgoing edge in $A(C)$ restricted to the non-boundary particles must have an outgoing edge to a boundary particle (otherwise it is a boundary particle itself), $A(C)$ is a forest or a ring of trees. The second statement of the lemma follows from the fact that every boundary particle must be connected to the object or a retired particle. □

Finally, we investigate the structure formed by the idle particles.

Lemma 12. *At any time, every connected component of idle particles is connected to at least one non-idle particle or the object.*

Proof. Initially, the lemma holds by the definition of a valid instance. Suppose that the lemma holds at time t and consider a connected component of idle particles. If one of the idle particles in the component is activated, it may either stay idle or change to an active particle, but in both cases the lemma holds at time $t + 1$. If a retired particle that is connected to the component is activated, it does not move. If a follower or root particle that is connected to the component is activated, that particle cannot contract outside of a handover with another follower or root particle, which implies that no node occupied by it is given up by the active particles. So in any of these cases, the connected component of idle particle remains connected to a non-idle particle. Therefore, the lemma holds at time $t + 1$. □

The following corollary is consequence of the previous three lemmas.

Corollary 1. *At any time, $V(P) \cup V(O)$ forms a single connected component.*

Lemma 13. *At any time before the first particle retires, in every connected component G of $A(C)$, the number of expanded boundary particles in G plus the number of complaint flags in G is equal to the number of non-boundary particles in G.*

Proof. Initially, the lemma holds trivially. Suppose the lemma holds at time t and consider the next activation of a particle. We only discuss relevant cases. If an idle particle becomes a non-boundary particle (i.e., it is not connected to the object but joins a connected component), it also generates a complaint flag. So both the number of non-boundary particles and the number of complaint flags increases by one for the component the particle joins. If a non-boundary particle expands as part of a handover with a boundary particle, both the number of expanded boundary particles and the number of non-boundary particles decrease by one for the component. If a boundary particle expands as part of a handover, that handover must be with another boundary particle, so the number of expanded boundary particles remains unchanged for that component. Since by our assumption there is no retired particle, all boundary particles are in layer 1. Hence, for a boundary particle to expand outside of a handover, it has to consume a complaint flag. This increases the number of expanded boundary particles by one and decreases the number of complaint flags by one. Finally, an expansion of a boundary particle outside of a handover can connect two components of $A(C)$. Since the equation given in the lemma holds for each of these components individually, it also holds for the newly built component. □

6.3.2 Liveness

We say that the particle system *makes progress* if (i) an idle particle becomes active, or (ii) a movement (i.e., an expansion, handover, or contraction) is executed, or (iii) an active particle retires. In the following, we always assume that we have a fair activation sequence for the particles.

Before we show under which circumstances our particle system eventually makes progress, we first establish some lemmas on how particles behave during the execution of our algorithm.

Lemma 14. *Eventually, every idle particle becomes active.*

Proof. As long as an idle particle exists, there is always an idle particle p that is connected to a non-idle particle or the object according to Lemma 12. The next time p is activated p becomes active according to Algorithm 7. Therefore, eventually all particles become active. □

The following statement shows that even though super-roots can be followers, they will become a boundary particle the next time they are activated.

Lemma 15. *In every tree of $A(C)$, every boundary particle in the follower state enters a root state the next time it is activated. In particular, every super-root in $A(C)$ will enter the root state the next time it is activated.*

Proof. Let p be a follower boundary particle. By definition p must have a retired particle or the object in its neighborhood. Therefore, p immediately becomes a root particle once it is activated according to Algorithm 7. □

Furthermore, the following lemma provides a relation between the movement of super-roots and the availability of complaint flags.

Lemma 16. *For every tree of $A(C)$ with a contracted super-root p and at least one complaint flag, p will eventually retire or expand to $p.dir$, thereby consuming a complaint flag, and after the expansion p may cease to be a super-root.*

Proof. If p is not a root, it becomes one the next time it is activated according to Lemma 15. Therefore, assume p is a root. If there is a retired particle in $p.dir$, p retires and ceases to be a super-root. If the node in $p.dir$ is unoccupied, p can potentially expand. According to Algorithm 9, complaint flags are forwarded along the tree of p towards p. Once the flag reaches p, it will expand, thereby consuming the flag. If p expands, it might have an active particle in its movement direction and thus ceases to be a super-root. □

Next, we prove the statement that expanded particles will not starve, i.e., they will eventually contract.

Lemma 17. *Eventually, every expanded particle contracts.*

Proof. Consider an expanded particle p in a configuration C. By Lemma 14 we can assume w.l.o.g. that all particles in C are active or retired. If there is no particle q with either $q.parent = p$ or p occupying the node in $q.dir$, then p can contract once it is activated. If such a q exists and it is contracted, p contracts in a handover (see Algorithm 8). If q exists and is expanded, we consider the tree of $A(C)$ that p is part of. Consider a subpath in this tree that starts in p, i.e., $(v_1, v_2, \ldots, v_k)$ such that v_1, v_2 are occupied by p and v_k is a node that does not have an incoming edge in $A(C)$. Let v_i be the first node of this path that is occupied by a contracted particle. If all particles are expanded, then clearly the last particle occupying v_{k-1}, v_k eventually contracts and we can set v_i to v_{k-1}. Since v_i is contracted it eventually performs a handover with the particle occupying v_{i-2}, v_{i-1}. Now we can move backwards along $(v_1, v_2, \ldots, v_{i-1})$ and it is guaranteed that a contracted particle eventually performs

a handover with the expanded particle occupying the two nodes before it on the path. So eventually q is contracted, eventually performs a handover with p and the statement holds. □

In the following two lemmas we will specifically consider the case that $B \leq n$, i.e., the particles can coat at least one layer around the object.

Lemma 18. *If $B \leq n$, layer 1 is completely filled with contracted particles eventually.*

Proof. Consider a configuration C such that layer 1 is not completely filled by contracted particles. Note that in this case the leader election cannot have succeeded yet, which means that a leader cannot be elected, and therefore no particle can be retired in configuration C. So by Lemma 14 we can assume w.l.o.g. that all particles in configuration C are active.

Since layer 1 is not completely filled by contracted particles, there is either at least one unoccupied node v on layer 1 or all nodes are occupied, but there is at least one expanded particle on layer 1. We show that in both cases a follower will move to layer 1, thereby filling up the layer until all particles are contracted. In the first case, let p be the super-root of a tree in $A(C)$ that still has non-boundary particles, let $(p_0 = p, p_1, \ldots, p_k)$ be the boundary particles of the tree such that p_{i-1} occupies the node in $p_i.dir$ and let q be the non-boundary particle in the tree that is adjacent to some $p_j \in (p_0, \ldots, p_k)$ such that j is minimal. If a particle p_i in $(p_0, \ldots, p_j = q.parent)$ is expanded, it eventually contracts (Lemma 17) by a handover with p_{i+1}, and by consecutive handovers all particles in $(p_{i+1}, \ldots, p_j)$ eventually expand and contract until the particle $p_j = q.parent$ expands. According to Algorithm 8, p_j performs a handover with q. Therefore, the number of particles on layer 1 has increased. If all particles in $(p_0, \ldots, q.parent)$ are contracted, then by Lemma 13 a complaint flag still exists in the tree. Eventually, p expands by Lemma 16. Consequently, we are back in

the former case that a particle in $(p_0, \ldots, q.parent)$ is expanded. In the second case, let p' be an expanded boundary particle and let q' be the non-boundary particle with the shortest path in $A(C)$ to p'. By a similar argument as for the first case, particles on layer 1 perform handovers (starting with p') until eventually the node in $q'.parent$ is occupied by a tail. Again, q' eventually performs a handover and the number of particles on layer 1 has increased. □

As a direct consequence, we can show the following.

Lemma 19. *If $B \leq n$, a leader is elected in layer 1 eventually.*

Proof. According to Lemma 18 layer 1 is eventually filled with contracted particles. Leader Election successfully elects a leader node according to [36]. The contracted particle p occupying the leader node forwards the *CHK* flag and eventually receives it back, since all particles are contracted. Therefore, p becomes a leader. □

Now we are ready to prove the two major statements of this subsection that define two conditions for system progress.

Lemma 20. *If all particles are non-retired and there is either a complaint flag or an expanded particle, the system eventually makes progress.*

Proof. If there is an idle particle, progress is ensured by Lemma 14. If an active particle is expanded Lemma 17 guarantees progress. Finally, in the last case all particles are active, none of them is expanded and there is a complaint flag. If layer 1 is completely filled, a leader is elected according to Lemma 19 and as a direct consequence the active particles on layer 1 eventually retire, guaranteeing progress. If layer 1 is not completely filled, there exists at least one tree of $A(C)$ with a contracted super-root p that has an unoccupied node in $p.dir$ and at least one complaint flag. Therefore, progress is ensured by Lemma 16. □

Lemma 21. *If there is at least one retired particle and one active particle, the system eventually makes progress.*

Proof. Again, if there is an idle particle, progress is ensured by Lemma 14. Moreover, note that since there is at least one retired particle, we can conclude that leader election has been successful (since the first particle that retires is a leader particle) and therefore layer 1 has to be completely filled with contracted particles. If there is still a non-retired particle on layer 1, it eventually retires according to the Algorithm, guaranteeing progress.

So suppose that all particles in layer 1 are retired. We distinguish between the following cases: (i) there exists at least one super-root, (ii) no super-root exists, but there is an expanded particle, and (iii) no super-root exists and all particles are contracted. In case (i), Lemma 15 guarantees that a super-root will eventually enter root state, and therefore it will eventually either expand (if $p.dir$ is unoccupied) or retire (since $p.dir$ is occupied by a retired particle). In case (ii), the particle contracts according to Lemma 17. In case (iii) $A(C)$ forms a ring of trees, which can only happen if all boundary particles completely occupy a single layer, so there is an active particle that occupies the node adjacent to the marker edge. Since it is contracted by assumption, it retires upon activation. Therefore, in all three cases the system eventually makes progress. □

6.3.3 Termination

Finally, we show that the algorithm eventually terminates in a legal configuration, i.e., a configuration in which the coating problem is solved. For the termination we need the following two lemmas.

Lemma 22. *The number of times an idle particle is transformed into an active one*

and an active particle is transformed into a retired one is bounded by $\mathcal{O}(n)$.

Proof. From our algorithm it immediately follows that every idle particle can only be transformed once into an active particle, and every active particle can only be transformed once into a retired particle. Moreover, a non-idle particle can never become idle again, and a retired particle can never become non-retired again, which proves the lemma. □

Lemma 23. *The overall number of expansions, handovers, and contractions is bounded by $\mathcal{O}(n^2)$.*

Proof. We will need the following fact, which immediately follows from our algorithm.

Fact 1. *Only a super-root of $A(C)$ can expand to a non-occupied node, and every such expansion triggers a sequence of handovers, followed by a contraction, in which every particle participates at most twice.*

Consider any particle p. Note that only an active particle performs a movement. Let C be the first configuration in which p becomes active. If it is a non-boundary particle (i.e., a follower), then consider the directed path in $A(C)$ from the head of p to the super-root r of its tree or the first particle r belonging to the ring in the ring of trees. Such a path must exist due to Lemma 11. Let $P = (v_0, v_1, \ldots, v_m)$ be a node sequence covered by this path where v_0 is the head of p in C and v_m is the first node along that path with the object or a retired particle in its neighborhood. Note that by Lemma 11 such a node sequence is well-defined since v_m must at latest be a node occupied by r. According to Algorithm 7, p attempts to follow P by sequentially expanding into the nodes $v_0, v_1, \ldots, v_m$. At latest, p will become a boundary particle once it reaches v_m. Up to this point, p has traveled along a path of length at most $2n$, and therefore, the number of movements p executes as a follower is $\mathcal{O}(n)$.

Now suppose p is a boundary particle. Let C be the configuration in which p becomes a boundary particle and let $\ell = p.layer$. Suppose that $\ell = 1$. From our algorithm we know that at most n complaint flags are generated by the particles, and therefore by Lemma 16, there are at most n expansions in level 1 (the rest are handovers or contractions). Hence, it follows from Fact 1 that p can only move $\mathcal{O}(n)$ times as a boundary particle.

Next consider the case that $\ell > 1$. Here we will need the following well-known fact.

Fact 2. *Let B_i be the length of layer i. For every i and every valid instance (P, O) allowing O to be coated by i layers it holds that $B_i = B_0 + 6i$.*

If $\ell = 2$, there must be a retired particle in layer 1, and since the leader is the first retired particle, Lemmas 18 and 19 imply that level $\ell - 1$ is completely filled with contracted particles. So p can only move along nodes of layer ℓ. Since $B_{\ell-1} \leq n$, it follows from Fact 2 that $B_\ell \leq n + 6$. As long as not all particles in level $\ell - 1$ are retired, p cannot move beyond the marker node in level ℓ. So p either becomes retired before reaching the marker node, or if it reaches the marker node, it has to wait there till all particles in level $\ell - 1$ are retired, which causes the retirement of p. Therefore, p moves along at most $n + 6$ nodes. If $\ell > 2$, we know from Lemma 10 that level $\ell - 2$ is completely filled with contracted particles. Since $B_{\ell-2} \leq n$ and $B_\ell = B_{\ell-2} + 12$, it follows that $B_\ell \leq n + 12$. Hence, p will move along at most $n + 12$ nodes in level ℓ before becoming retired or moving to level $\ell - 1$, and p will move along at most $n + 6$ further nodes in level $\ell - 1$ before retiring.

Thus, in any case, p performs at most $\mathcal{O}(n)$ movements as a boundary particle. Therefore, the number of movements any particle in the system performs is $\mathcal{O}(n)$, which concludes the lemma. □

Lemmas 22 and 23 imply that the system can only make progress $\mathcal{O}(n^2)$ many times. Hence, eventually our system reaches a configuration in which it no longer makes progress, so the system terminates. It remains to show that when the algorithm terminates, it is in a legal configuration, i.e., the algorithm solves the coating problem.

Theorem 10. *Our coating algorithm terminates in a legal configuration.*

Proof. From the conditions of Lemmas 20 and 21 we know that the following facts must both be true when the algorithm terminates:

1. At least one particle is retired or there is neither a complaint flag nor an expanded particle in the system (Lemma 20).

2. Either all particles are retired or all particles are active (Lemma 21).

First suppose that all particle are retired. Then it follows from Lemma 10 that the configuration is legal. Next, suppose that all particles are active and neither a complaint flag nor an expanded particle is left in the system. Then Lemma 13 implies that there cannot be any non-boundary any more, so all active particles must be boundary particles. If there is at least one boundary particle in layer $\ell > 1$, then there must be at least one retired particle, contradicting our assumption. So all boundary particles must be in layer 1, and since there are no more complaint flags and all boundary particles are contracted, also in this case our algorithm has reached a legal configuration, which proves our theorem. □

Recall that the *work* performed by an algorithm is defined as the number of movements (expansions, handovers, and contractions) of the particles till it terminates. Lemma 23 implies that the work performed by our algorithm is $\mathcal{O}(n^2)$. Interestingly, this is also the best bound one can achieve in the worst-case for the coating problem.

Lemma 24. *The worst-case work required by any algorithm to solve the Universal Object Coating problem is* $\Omega(n^2)$.

Proof. Consider the configuration of particles which is a straight line connected to the object from an endpoint of the line. A particle with distance $i \geq 1$ to the object needs at least $2(i-1-\lceil \frac{i-1}{B} \rceil)$ movements to become contracted on its final layer. Therefore, any algorithm requires at least $2\sum_{i=1}^{n-1}(i-1-\lceil \frac{i-1}{B} \rceil) \geq \sum_{i=1}^{n-1}(i-1-(\frac{i}{B})) = \Omega(n^2)$ work assuming $B \geq 2$. □

Hence, we get:

Theorem 11. *Our algorithm requires worst-case optimal work* $\Theta(n^2)$.

6.4 Applications

In this section, we present other coating scenarios and applications of our universal coating algorithm. Our algorithm can be easily extended to also handle the case when one would like to cover only a certain portion of the object surface. More concretely, assume that one would like to cover the portion of the object surface delimited by two *endpoint nodes*. Basically in that case, the algorithm can be modified slightly so that the particles that eventually reach one of the endpoints of the surface segment retire and become *endpoint markers*. The position of endpoint marker particles will be propagated to higher layers, as necessary, such that the delimited portion of the object is evenly coated.

Once the first layer is formed and a leader is elected (implying that $B \leq n$), one can trivially determine (i) whether the number of particles in the system is greater than or equal to the size of the object boundary, or (ii) whether the object O is convex; one could also potentially address other applications that involve aggregating some (constant-size) collective data over the boundary of the object O. Once all particles

in layer 1 retire, a leader will emerge and that leader can initiate the respective application. For the first application, all particles may initially assume that $B > n$. Once a leader is elected, it informs all other particles that $B \leq n$. For the convexity testing, the leader particle can generate a token that traverses the boundary in CW direction: If the token ever makes a left turn (i.e., it traverses two consecutive edges on the boundary at an outer angle of less than 180°), then the object is not convex; otherwise the object is convex.

6.5 Conclusion

We presented a universal coating algorithm for programmable matter using worst-case optimal work. In the next section we bound the parallel runtime of our algorithm and investigate its runtime competitiveness, i.e., how does its runtime compare to the best possible runtime for any given instance. Moreover, it would be interesting to implement the algorithm and evaluate its performance either via simulations or hopefully at some point even via experiments with real programmable matter.

Chapter 7

ON THE RUNTIME OF UNIVERSAL COATING

7.1 Introduction

We continue the study of coating problems in the context of self-organizing programmable matter consisting of simple computational elements, called particles, that can establish and release bonds and can actively move in a self-organized way using the geometric version of the amoebot model presented in Section 2 (as well as [26, 30]). In doing so, we proceed to investigate the runtime analysis of our Universal Coating algorithm, introduced in Section 6 and [34]. We first show that coating problems do not only have a (trivial) linear lower bound on the runtime, but that there is also a linear lower bound on the competitive gap between the runtime of fully local coating algorithms and coating algorithms that rely on global information. We then investigate the worst-case time complexity of our Universal Coating algorithm and show that it terminates within a linear number of rounds with high probability (w.h.p.)[1], which implies that our algorithm is optimal in terms of worst-case runtime and also in a competitive sense. Moreover, our simulation results show that in practice the competitive ratio of our algorithm is often better than linear.

7.1.1 Our Contributions

We now continue the analysis of the *Universal Coating algorithm* introduced in Section 6. As our main contribution in this chapter, we investigate the runtime of

[1]By *with high probability*, we mean with probability at least $1 - 1/n^c$, where n is the number of particles in the system and $c > 0$ is a constant.

our algorithm and prove that our algorithm terminates within a *linear number of rounds* with high probability. We also present a matching linear lower bound for local-control coating algorithms that holds with high probability. We use this lower bound to show a *linear lower bound on the competitive gap* between fully local coating algorithms and coating algorithms that rely on global information, which implies that our algorithm is also optimal in a competitive sense. We then present some simulation results demonstrating that in practice the competitive ratio of our algorithm is often much better than linear.

7.1.2 Overview

In Section 7.2, we present a brief overview of the algorithm. We then present a comprehensive formal runtime analysis of our algorithm, by first presenting some lower bounds on the competitive ratio of any local-control algorithm in Section 7.3, and then proving that our algorithm has a runtime of $\mathcal{O}(n)$ rounds w.h.p. in Section 7.4, which matches our lower bounds. This study appears in [31].

7.2 Brief Overview of the Universal Coating Algorithm

In this section we summarize the Universal Coating algorithm introduced in Section 6.2 and [34]. This algorithm is constructed by combining a number of asynchronous coating primitives, which are integrated seamlessly without any underlying synchronization.

7.2.1 Overview of Coating Primitives

The ***spanning forest primitive*** organizes the particles in a spanning forest $\mathcal{F}$, which creates a straightforward mechanism for particles to move while preserving connectivity (see Section 6.2.2 and [26, 32] for details). Initially, all particles are

idle. A particle p touching the object changes its state to *root*. For any other idle particle p we use the rule that once p sees a root or a follower in its neighborhood, it stores the direction to one of them in $p.parent$, changes its state to *follower*, and generates a complaint flag. Follower particles use handovers to follow their parents and update $p.parent$ as they move so that it always points to the same parent (resp. the follower that took over the role of p's parent q because of a handover with q). In this way, the trees formed by the parent relations stay connected, only use positions they have covered before, and do not mix with other trees. Roots p use the flag $p.dir$ to determine their movement direction. As a root p moves, it updates $p.dir$ so that it always points to the next position of a clockwise movement around the object. For any particle p, we call the particle occupying the position that $p.parent$ resp. $p.dir$ points to the *predecessor* of p. If a root particle does not have a predecessor, we call it a *super-root*.

The ***complaint-based coating primitive*** is used for the coating of the first layer. This primitive controls the coating of the first layer, by expanding the coating of the first layer while there is still room and there are still particles not yet touching the object Each time a particle p holding at least one complaint flag is activated, it forwards one to its predecessor as long as that predecessor does not hold more than two complaint flags. Allowing each particle to hold up to two complaint flags has two reasons: it ensures that a constant size memory is sufficient for storing the complaint flags and that the flags quickly move forward to the super-roots. A contracted super-root p only expands to $p.dir$ if it holds at least one complaint flag, and when it expands, it consumes one of these complaint flags. All other roots p move towards $p.dir$ whenever possible (i.e., no complaint flags are needed for that) by performing a handover with a predecessor (which must be another root) or a successor (which is a root or a follower of its tree), with preference given to a follower so that one more

particle reaches layer 1. As we will see, these rules ensure that whenever there are particles in the system that are not yet at layer 1, eventually one of these particles will move to layer 1, unless layer 1 is already completely filled with contracted particles.

The ***node-based leader election primitive*** runs during the complaint-based coating primitive to elect a position along layer 1 as the leader position. This algorithm is similar to the algorithm presented in [26] with the difference that leader candidates are associated with positions instead of particles (which is important because in our case particles may still move during the leader election primitive) as we presented in [34]. The primitive only terminates once all positions in layer 1 are occupied. Once the leader position is determined, and there are no more followers or all positions in layer 1 are filled by contracted particles, then whatever particle currently covers that position becomes the *leader*. If the primitive does not terminate (which only happens if $n < B_1$ and layer 1 is never completely covered), then the complaint flags ensure that the super-roots eventually stop, which eventually results in a stable legal coating. Given that a particle becomes the leader, that leader becomes a marker particle that marks a neighboring position at the next higher layer as a *marked position* and changes to the *retired* state. The marked position determines the point at which root particles should align in the next higher layer. Once a contracted root p has a retired particle in the direction $p.dir$, it retires as well, which causes the particles in layer 1 to change to the *retired* state in a counter-clockwise order. Also, the general layering primitive becomes active, which builds subsequent layers until there are no longer followers in the system.

The ***general layering primitive*** allows each layer i to form only after layer $i-1$ has been completed, for $i \geq 2$. In this primitive, whenever a follower is connected to a retired particle, it changes to the root state. Root particles continue to move along positions of their layer in a clockwise (if the layer number is odd) or counter-

clockwise (if the layer number is even) direction till they reach the marked position in that layer, or a retired particle in that layer, or a previously empty position of a lower layer (which causes them to change direction). Complaint flags are no longer needed to move to empty spots. Followers follow their parents as before. A contracted root particle may retire due to the following reasons: (i) it is located at the marked position and the marker particle in the lower layer tells it that all particles in that layer are already retired (which it can determine locally), or (ii) it has a retired particle in the direction of its layer. Once a particle at a marked position retires, it becomes a marker particle and marks a neighboring position at the next higher layer as a marked position.

7.3 Lower Bounds

Recall that a *round* is over once every particle in P has been activated at least once. The *runtime* $T_{\mathcal{A}}(P, O)$ of a coating algorithm $\mathcal{A}$ is defined as the worst-case number of rounds (over all sequences of particle activations) required for $\mathcal{A}$ to solve the coating problem (P, O). Certainly, there are instances (P, O) where every coating algorithm has a runtime of $\Omega(n)$ (see Lemma 25), though there are also many other instances where the coating problem can be solved much faster. Since a worst-case runtime of $\Omega(n)$ is fairly large and therefore not very helpful to distinguish between different coating algorithms, we intend to study the runtime of coating algorithms relative to the best possible runtime.

Lemma 25. *The worst-case runtime required by any local-control algorithm to solve the universal coating problem is* $\Omega(n)$.

Proof. Assume the particles $p_1, \ldots, p_n$ form a single line of n particles connected to the surface via p_1 (Figure 7.1). Suppose $B_1 > n$. Since $d(p_n, O) = n$, it will take $\Omega(n)$ rounds in the worst-case (requiring $\Theta(n)$ movements) until p_n touches the object's

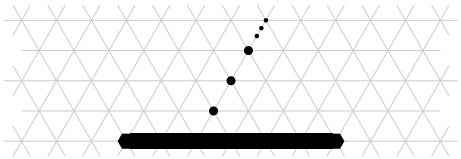

Figure 7.1: Worst-case configuration concerning number of rounds. There are n particles (black dots) in a line connected to the surface via a single particle p_1.

surface. This worst-case can happen, for example, if p_n performs no more than one movement (either an expansion or a contraction) per round. □

Unfortunately, a large lower bound also holds for the competitiveness of any local-control algorithm. A coating algorithm $\mathcal{A}$ is called *c-competitive* if for any valid instance (P, O),

$$\mathrm{E}[T_{\mathcal{A}}(P, O)] \leq c \cdot \mathrm{OPT}(P, O) + K$$

where $\mathrm{OPT}(P, O)$ is the minimum runtime needed to solve the coating problem (P, O) and K is a value independent of (P, O).

Theorem 12. *Any local-control algorithm that solves the universal coating problem has a competitive ratio of* $\Omega(n)$.

Proof. We construct an instance of the coating problem (P, O) which can be solved by an optimal algorithm in $\mathcal{O}(1)$ rounds, but requires any local-control algorithm $\Omega(n)$ times longer. Let O be a straight line of arbitrary (finite) length, and let P be a set of particles which entirely occupy layer 1, with the exception of one missing particle below O equidistant from its sides and one additional particle above O in layer 2 equidistant from its sides (see Figure 7.2).

An optimal algorithm could move the particles to solve the coating problem for the given example in $\mathcal{O}(1)$ rounds, as in Figure 7.3. Note that the optimal algorithm always maintains the connectivity of the particle system, so its runtime is valid even under the constraint that any connected component of particles must stay connected.

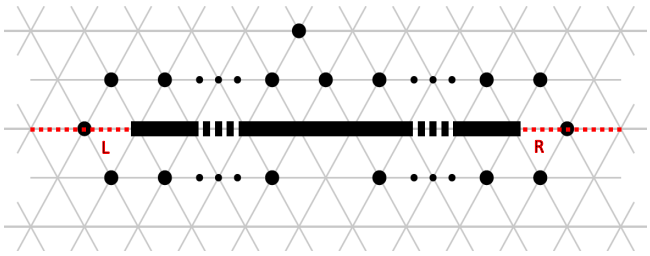

Figure 7.2: The object occupies a straight line in G_{eqt}. The particles are all contracted and occupy the positions around the object, with the exception that there is one unoccupied node below the object and one extra particle above the object. Borders L and R are shown as red lines.

However, for our local-control algorithms we allow particles to disconnect from the rest of the system.

Now consider an arbitrary local-control algorithm A for the coating problem. Given a round r, we define the *imbalance* $\phi_L(r)$ at border L as the net number of particles that have crossed L from the top of O to the bottom until round r; similarly, the imbalance $\phi_R(r)$ at border R is defined to be the net number of particles that have crossed R from the bottom of O to the top until round r.

Certainly, there is an activation sequence in which information and particles can only travel a distance of up to $n/4$ nodes towards L or R within the first $n/4$ rounds. Hence, for any $r \leq n/4$, the probability distributions of $\phi_L(r)$ and $\phi_R(r)$ are independent of each other. Additionally, particles up to a distance of $n/4$ from L and R cannot distinguish between which border they are closer to, since the position of the gap is equidistant from the borders. This symmetry also implies that $\Pr[\phi_L(r) = k] = \Pr[\phi_R(r) = k]$ for any integer k. Let us focus on round $r = n/4$. We distinguish between the following cases.

Case 1: $\phi_L(n/4) = \phi_R(n/4)$. Then there are more particles than positions in layer 1 above O, so the coating problem cannot be solved yet.

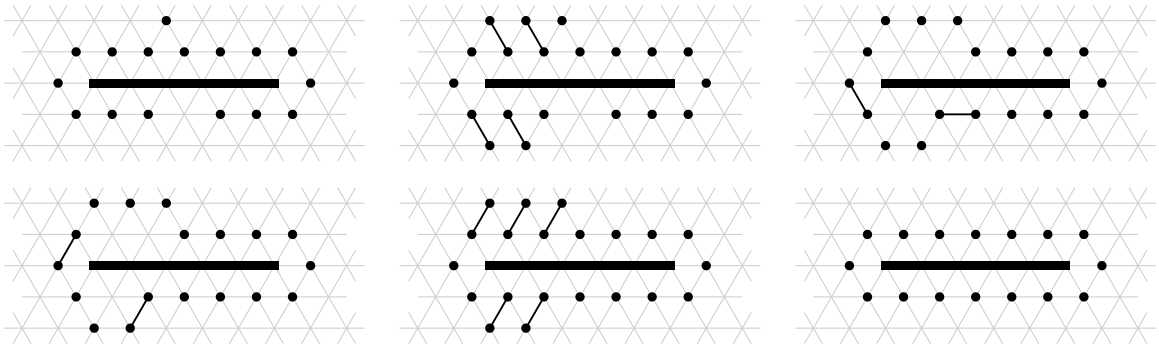

Figure 7.3: Each subfigure represents the configuration of the system at the beginning of a round, and are ordered from left to right, top to bottom. After 5 rounds (i.e., at the beginning of the sixth round) the object is coated. Note that the implied algorithm can be adapted to any length of the object and always requires only 5 rounds to solve the coating problem.

Case 2: $\phi_L(n/4) \neq \phi_R(n/4)$. From our insights above we know that for any two values k_1 and k_2, $\Pr[\phi_L(n/4) = k_1 \text{ and } \phi_R(n/4) = k_2] = \Pr[\phi_L(n/4) = k_2 \text{ and } \phi_R(n/4) = k_1]$. Hence, the cumulative probability of all outcomes where $\phi_L(n/4) < \phi_R(n/4)$ is equal to the cumulative probability of all outcomes where $\phi_L(n/4) > \phi_R(n/4)$. If $\phi_L(n/4) < \phi_R(n/4)$, then there are again more particles than positions in layer 1 above O, so the coating problem cannot be solved yet.

Thus, the probability that $\mathcal{A}$ has not solved the coating problem after $n/4$ rounds is at least $1/2$, and therefore $\mathrm{E}[T_{\mathcal{A}}(P, O)] \geq 1/2 \cdot n/4 = n/8$. Since, on the other hand, $\mathrm{OPT} = \mathcal{O}(1)$, we have established a linear competitive ratio. □

Therefore, even the competitive ratio can be very high in the worst case. We will revisit the notion of competitiveness in Section 7.5.

7.4 Worst-Case Number of Rounds

In this section, we show that our algorithm solves the coating problem within a linear number of rounds w.h.p. We start with some basic notation in Section 7.4.1. Section 7.4.2 presents a simpler synchronous parallel model for particle activations that we can use to analyze the worst-case number of rounds. Section 7.4.3 presents the analysis of the number of rounds required to coat the first layer. Finally, in Section 7.4.4, we analyze the number of rounds required to complete all other coating layers, once layer 1 has been completed.

7.4.1 Preliminaries

We start with some notation. Recall that B_i denotes the number of nodes of G_{eqt} at distance i from object O (i.e., the number of nodes in layer i). Let N be the the layer number of the final layer for n particles (i.e., N satisfies $\sum_{j=1}^{N-1} B_j < n \leq \sum_{j=1}^{N} B_j$). Layer i is said to be *complete* if every node in layer i is occupied by a contracted retired particle (for $i < N$), or if all particles have reached their final position, are contracted, and never move again (for $i = N$).

Given a configuration C, we define a directed graph $A(C)$ over all nodes in G_{eqt} occupied by *active* (follower or root) particles in C. For every expanded active particle p in C, $A(C)$ contains a directed edge from the tail to the head of p. For every follower p, $A(C)$ has a directed edge from the head of p to $p.parent$. For the purposes of constructing $A(C)$, we also define parents for root particles: a root particle p sets $p.parent$ to be the active particle q occupying the node in direction $p.dir$ once p has performed its first handover expansion with q. For every root particle p, $A(C)$ has a directed edge from the head of p to $p.parent$, if it exists. Certainly, since every node has at most one outgoing edge in $A(C)$, the nodes of $A(C)$ form either a collection

of disjoint trees or a ring of trees. A ring of trees may occur in any layer, but only temporarily; the leader election primitive ensures that a leader emerges and retires in layer 1 and marker particles emerge and retire in higher layers, causing the ring in $A(C)$ to break. The super-roots defined in Section 6.2.2 correspond to the roots of the trees in $A(C)$.

A *movement* executed by a particle p can be either a *sole contraction* in which p contracts and leaves a node unoccupied, a *sole expansion* in which p expands into an adjacent unoccupied node, a *handover contraction with* p' in which p contracts and forces its contracted neighbor p' to expand into the node it vacates, or a *handover expansion with* p' in which p expands into a node currently occupied by its expanded neighbor p', forcing p' to contract.

7.4.2 From asynchronous to parallel schedules

In this section, we show that instead of analyzing our algorithm for asynchronous activations of particles, it suffices to consider a much simpler model of parallel activations of particles. Define a *movement schedule* to be a sequence of particle system configurations $(C_0, \ldots, C_t)$.

Definition 1: A movement schedule $(C_0, \ldots, C_t)$ is called a *parallel schedule* if each C_i is a valid configuration of a connected particle system (i.e., each particle is either expanded or contracted, and every node of G_{eqt} is occupied by at most one particle) and for every $i \geq 0$, C_{i+1} is reached from C_i such that for every particle p one of the following properties holds:

1. p occupies the same node(s) in C_i and C_{i+1},
2. p expands into an adjacent node that was empty in C_i,
3. p contracts, leaving the node occupied by its tail empty in C_{i+1}, or

4. p is part of a handover with a neighboring particle p'.

While these properties allow at most one contraction or expansion per particle in moving from C_i to C_{i+1}, multiple particles may move in this time.

Consider an arbitrary fair asynchronous activation sequence A for a particle system, and let $C_i^{(A)}$, for $0 \leq i \leq t$, be the particle system configuration at the end of asynchronous round i in A if each particle moves according to Algorithm 7. A *forest schedule* $\mathcal{S} = (A, (C_0, \ldots, C_t))$ is a parallel schedule $(C_0, \ldots, C_t)$ with the property that $A(C_0)$ is a forest of one or more trees, and each particle p follows the unique path P_p which it would have followed according to A, starting from its position in C_0. This implies that $A(C_i)$ remains a forest of trees for every $1 \leq i \leq t$. A forest schedule is said to be *greedy* if all particles perform movements according to Definition 7.4.2 in the direction of their unique paths whenever possible.

We begin our analysis with a result that is critical to both describing configurations of particles in greedy forest schedules and quantifying the amount of progress greedy forest schedules make over time. Specifically, we show that if a forest's configuration is "well-behaved" at the start, then it remains so throughout its greedy forest schedule, guaranteeing that progress is made once every two configurations.

Lemma 26. *Given any fair asynchronous activation sequence A, consider any greedy forest schedule $(A, (C_0, \ldots, C_t))$. If every expanded parent in C_0 has at least one contracted child, then every expanded parent in C_i also has at least one contracted child, for $1 \leq i \leq t$.*

Proof. Suppose to the contrary that C_i is the first configuration that contains an expanded parent p which has all expanded children. We consider all possible expanded and contracted states of p and its children in C_{i-1} and show that none of them can result in p and its children all being expanded in C_i. First suppose p is expanded in

C_{i-1}; then by supposition, p has a contracted child q. By Definition 7.4.2, q cannot perform any movements with its children (if they exist), so p performs a handover contraction with q, yielding p contracted in C_i, a contradiction. So suppose p is contracted in C_{i-1}. We know p will perform either a handover with its parent or a sole expansion in direction $p.dir$ since it is expanded in C_i by supposition. Thus, any child of p in C_{i-1} — say q — does not execute a movement with p in moving from C_{i-1} to C_i. Instead, if q is contracted in C_{i-1} then it remains contracted in C_i since it is only permitted to perform a handover with its unique parent p; otherwise, if q is expanded, it performs either a sole contraction if it has no children or a handover with one of its contracted children, which it must have by supposition. In either case, p has a contracted child in C_i, a contradiction.

As a final observation, two trees of the forest may "merge" when the super-root s of one tree performs a sole expansion into an unoccupied node adjacent to a particle q of another tree. However, since s is a root and thus only defines q as its parent after performing a handover expansion with it, the lemma holds in this case as well. □

For any particle p in a configuration C of a forest schedule, we define its *head distance* $d_h(p, C)$ (resp., *tail distance* $d_t(p, C)$) to be the number of edges along P_p from the head (resp., tail) of p to the end of P_p. Depending on whether p is contracted or expanded, we have $d_h(p, C) \in \{d_t(p, C), d_t(p, C)-1\}$. For any two configurations C and C' and any particle p, we say that C *dominates* C' *w.r.t.* p, denoted $C(p) \succeq C'(p)$, if and only if $d_h(p, C) \leq d_h(p, C')$ and $d_t(p, C) \leq d_t(p, C')$. We say that C *dominates* C', denoted $C \succeq C'$, if and only if C dominates C' with respect to every particle. Then it holds:

Lemma 27. *Given any fair asynchronous activation sequence A which begins at an initial configuration $C_0^{(A)}$ in which every expanded parent has at least one contracted*

child, there is a greedy forest schedule $\mathcal{S} = (A, (C_0, \ldots, C_t))$ *with* $C_0 = C_0^{(A)}$ *such that* $C_i^{(A)} \succeq C_i$ *for all* $0 \leq i \leq t$.

Proof. We first introduce some supporting notation. Let $M(p) = p^{(1)}, p^{(2)}, \ldots$ be the sequence of movements p executes according to A. Let $M_i(p)$ denote the remaining sequence of movements in $M(p)$ after the forest schedule reaches C_i, and let $m_i(p)$ denote the first movement in $M_i(p)$.

Claim 3. *A greedy forest schedule* $\mathcal{S} = (A, (C_0, \ldots, C_t))$ *can be constructed from configuration* $C_0 = C_0^{(A)}$ *such that, for every* $0 \leq i \leq t$, *configuration* C_i *is obtained from* C_{i-1} *by executing only the movements of a greedily selected, mutually compatible subset of* $\{m_{i-1}(p) : p \in P\}$.

Proof. Argue by induction on i, the current configuration number. C_0 is trivially obtained, as it is the initial configuration. Assume by induction that the claim holds up to C_{i-1}. W.l.o.g. let $M_{i-1} = \{m_{i-1}(p_1), \ldots, m_{i-1}(p_k)\}$, for $k \leq n$, be the greedily selected, mutually compatible subset of movements that $\mathcal{S}$ performs in moving from C_{i-1} to C_i. Suppose to the contrary that a movement $m'(p) \notin M_{i-1}$ is executed by a particle $p \in P$. It is easily seen that $m'(p)$ cannot be $m_{i-1}(p)$; since $m_{i-1}(p)$ was excluded when M_{i-1} was greedily selected, it must be incompatible with one or more of the selected movements and thus cannot also be executed at this time. So $m'(p) \neq m_{i-1}(p)$, and we consider the following cases:

Case 1: $m_{i-1}(p)$ is a sole contraction. Then p is expanded and has no children in C_{i-1}, so we must have $m'(p) = m_{i-1}(p)$, since there are no other movements p could execute, a contradiction.

Case 2: $m_{i-1}(p)$ is a sole expansion. Then p is contracted and has no parent in C_{i-1}, so we must have $m'(p) = m_{i-1}(p)$, since there are no other movements p could execute, a contradiction.

Case 3: $m_{i-1}(p)$ is a handover contraction with q, one of its children. Then at some time in $\mathcal{S}$ before reaching C_{i-1}, q became a descendant of p; thus, q must also be a descendant of p in C_{i-1}. If q is not a child of p in C_{i-1}, there exists a particle $z \notin \{p, q\}$ such that q is a descendant of z, which is in turn a descendant of p. So in order for $m_{i-1}(p)$ to be a handover contraction with q, $M(z)$ must include actions which allow z to "bypass" its ancestor p, which is impossible. So q must be a child of p in C_{i-1}, and must be contracted at the time $m_{i-1}(p)$ is performed. If q is also contracted in C_{i-1}, then once again we must have $m'(p) = m_{i-1}(p)$. Otherwise, q is expanded in C_{i-1}, and must have become so before C_{i-1} was reached. But this yields a contradiction: since $\mathcal{S}$ is greedy, q would have contracted prior to this point by executing either a sole contraction if it has no children, or a handover contraction with a contracted child whose existence is guaranteed by Lemma 26, since every expanded parent in C_0 has a contracted child.

Case 4: $m_{i-1}(p)$ is a handover expansion with q, its unique parent. Then we must have that $m_{i-1}(q)$ is a handover contraction with p, and an argument analogous to that of Case 3 follows. □

We conclude by showing that each configuration of the greedy forest schedule $\mathcal{S}$ constructed according to the claim is dominated by its asynchronous counterpart. Argue by induction on i, the configuration number. Since $C_0 = C_0^{(A)}$, we have that $C_0^{(A)} \succeq C_0$. Assume by induction that for all rounds $0 \leq r \leq i-1$, we have $C_r^{(A)} \succeq C_r$. Consider any particle p. Since $\mathcal{S}$ is constructed using the exact set of movements p executes according to A and each time p moves it decreases either its head distance or tail distance by 1, it suffices to show that p has performed at most as many movements in $\mathcal{S}$ up to C_i as it has according to A up to $C_i^{(A)}$. If p does not perform a movement between C_{i-1} and C_i, we trivially have $C_i^{(A)}(p) \succeq C_i(p)$. Otherwise,

p performs movement $m_{i-1}(p)$ to obtain C_i from C_{i-1}. If p has already performed $m_{i-1}(p)$ according to A before reaching $C_{i-1}^{(A)}$, then clearly $C_i^{(A)}(p) \succeq C_i(p)$. Otherwise, $m_{i-1}(p)$ must be the next movement p is to perform according to A, since p has performed the same sequence of movements in the asynchronous execution as it has in $\mathcal{S}$ up to the respective rounds $i-1$, and thus has equal head and tail distances in C_{i-1} and $C_{i-1}^{(A)}$. It remains to show that p can indeed perform $m_{i-1}(p)$ between $C_{i-1}^{(A)}$ and $C_i^{(A)}$. If $m_{i-1}(p)$ is a sole expansion, then p is the super-root of its tree (in both C_{i-1} and $C_{i-1}^{(A)}$) and must also be able to expand in $C_{i-1}^{(A)}$. Similarly, if $m_{i-1}(p)$ is a sole contraction, then p has no children (in both C_{i-1} and $C_{i-1}^{(A)}$) and must be able to contract in $C_{i-1}^{(A)}$. If $m_{i-1}(p)$ is a handover expansion with its parent q, then q must be expanded in C_{i-1}. Parent q must also be expanded in $C_{i-1}^{(A)}$; otherwise $d_h(q, C_{i-1}^{(A)}) > d_h(q, C_{i-1})$, contradicting the induction hypothesis. An analogous argument holds if $m_{i-1}(p)$ is a handover contraction with one of its contracted children. Therefore, in any case we have $C_i^{(A)}(p) \succeq C_i(p)$, and since the choice of p was arbitrary, $C_i^{(A)} \succeq C_i$. □

We can show a similar dominance result when considering complaint flags.

Definition 2: A movement schedule $(C_0, \ldots, C_t)$ is called a *complaint-based parallel schedule* if each C_i is a valid configuration of a particle system in which every particle holds at most *one* complaint flag (rather than two, as described in Algorithm 9) and for every $i \geq 0$, C_{i+1} is reached from C_i such that for every particle p one of the following properties holds:

1. p does not hold a complaint flag and property 1, 3, or 4 of Definition 7.4.2 holds,

2. p holds a complaint flag f and expands into an adjacent node that was empty in C_i, consuming f,

3. p forwards a complaint flag f to a neighboring particle p' which either does not hold a complaint flag in C_i or is also forwarding its complaint flag.

A *complaint-based forest schedule* $\mathcal{S} = (A, (C_0, \ldots, C_t))$ has the same properties as a forest schedule, with the exception that $(C_0, \ldots, C_t)$ is a complaint-based parallel schedule as opposed to a parallel schedule. A complaint-based forest schedule is said to be *greedy* if all particles perform movements according to Definition 7.4.2 in the direction of their unique paths whenever possible.

We can now extend the dominance argument to hold with respect to *complaint distance* in addition to head and tail distances. For any particle p holding a complaint flag f in configuration C, we define its complaint distance $d_c(f, C)$ to be the number of edges along P_p from the node p occupies to the end of P_p. For any two configurations C and C' and any complaint flag f, we say that C *dominates* C' *w.r.t.* f, denoted $C(f) \succeq C'(f)$, if and only if $d_c(f, C) \leq d_c(f, C')$. Extending the previous notion of dominance, we say that C *dominates* C', denoted $C \succeq C'$, if and only if C dominates C' with respect to every particle and with respect to every complaint flag.

It is also possible to construct a greedy complaint-based forest schedule whose configurations are dominated by their asynchronous counterparts, as we did for greedy forest schedules in Lemma 27. Many of the details are the same, so as to avoid redundancy we highlight the differences here. The most obvious difference is the inclusion of complaint flags. Definition 7.4.2 restricts particles to holding at most one complaint flag at a time, where Algorithm 9 allows a capacity of two. This allows the asynchronous execution to not "fall behind" the parallel schedule in terms of forwarding complaint flags. Basically, Definition 7.4.2 allows a particle p holding a complaint flag f in the parallel schedule to forward f to its parent q even if q currently holds its own complaint flag, so long as q is also forwarding its flag at this time. The asynchronous execution does not have this luxury of synchronized actions,

so the mechanism of buffering up to two complaint flags at a time allows it to "mimic" the pipelining of forwarding complaint flags that is possible within one round of a complaint-based parallel schedule.

Another slight difference is that a contracted particle cannot expand into an empty adjacent node unless it holds a complaint flag to consume. However, this restriction reflects Algorithm 10, so once again the greedy complaint-based forest schedule can be constructed directly from the movements taken in the asynchronous execution. Moreover, since this restriction can only cause a contracted particle to remain contracted, the conditions of Lemma 26 are still upheld. Thus, we obtain the following lemma:

Lemma 28. *Given any fair asynchronous activation sequence A which begins at an initial configuration $C_0^{(A)}$ in which every expanded parent has at least one contracted child, there is a greedy complaint-based forest schedule $\mathcal{S} = (A, (C_0, \ldots, C_t))$ with $C_0 = C_0^{(A)}$ such that $C_i^{(A)} \succeq C_i$ for all $0 \leq i \leq t$.*

By Lemmas 27 and 28, once we have an upper bound for the time it takes a greedy forest schedule to reach a final configuration, we also have an upper bound for the number of rounds required by the asynchronous execution. Hence, the remainder of our proofs will serve to upper bound the number of parallel rounds any greedy forest schedule would require to solve the coating problem for a given valid instance (P, O), where $|P| = n$. Let $\mathcal{S}^* = (A, (C_0, \ldots, C_f))$ be such a greedy forest schedule, where C_0 is the initial configuration of the particle system P (of all contracted particles) and C_f is the final coating configuration.

In Sections 7.4.3 and 7.4.4, we will upper bound the number of parallel rounds required by $\mathcal{S}^*$ in the worst case to coat the first layer and higher layers, respectively. More specifically, we will bound the worst-case time it takes to complete a

layer i once layers $1, \ldots, i-1$ have been completed. For convenience, we will not differentiate between complaint-based and regular forest schedules in the following sections, since the same dominance result holds whether or not complaint flags are considered. To prove these bounds, we need one last definition: a *forest–path schedule* $\mathcal{S} = (A, (C_0, \ldots, C_t), L)$ is a forest schedule $(A, (C_0, \ldots, C_t))$ with the property that all the trees of $A(C_0)$ are rooted at a path $L = v_1 v_2 \cdots v_\ell \subseteq G_{\text{eqt}}$, and each particle p must traverse L in the same direction.

7.4.3 First layer: complaint-based coating and leader election

Our algorithm must first organize the particles using the spanning forest primitive, whose runtime is easily bounded:

Lemma 29. *Following the spanning forest primitive, the particles form a spanning forest within $\mathcal{O}(n)$ rounds.*

Proof. Initially all particles are idle. In each round any idle particle adjacent to the object, an active (follower or root) particle, or a retired particle becomes active. It then sets its parent flag if it is a follower, or becomes the root of a tree if it is adjacent to the object or a retired particle. In each round at least one particle becomes active, so — given n particles in the system — it will take $\mathcal{O}(n)$ rounds in the worst case until all particles join the spanning forest. □

For ease of presentation, we assume that the particle system is of sufficient size to fill the first layer (i.e., $B_1 \leq n$; the proofs can easily be extended to handle the case when $B_1 > n$); we also assume that the root of a tree also generates a complaint flag upon its activation (this assumption does not hurt our argument since it only increases the number of the flags generated in the system). Let $\mathcal{S}_1 = (A, (C_0, \ldots, C_{t_1}), L_1)$ be the greedy forest–path schedule where $(A, (C_0, \ldots, C_{t_1}))$ is a truncated version of $\mathcal{S}^*$,

C_{t_1} — for $t_1 \leq f$ — is the configuration in $\mathcal{S}^*$ in which layer 1 becomes complete, and L_1 is the path of nodes in layer 1. The following lemma shows that the algorithm makes steady progress towards completing layer 1.

Lemma 30. *Consider a round i of the greedy forest–path schedule $\mathcal{S}_1$, where $0 \leq i \leq t_1 - 2$. Then within the next two parallel rounds of $\mathcal{S}_1$, (i) at least one complaint flag is consumed, (ii) at least one more complaint flag reaches a particle in layer 1, (iii) all remaining complaint flags move one position closer to a super-root along L_1, or (iv) layer 1 is completely filled (possibly with some expanded particles).*

Proof. If layer 1 is filled, (iv) is satisfied; otherwise, there exists at least one super-root in $A(C_i)$. We consider several cases:

Case 1: There exists a super-root s in $A(C_i)$ which holds a complaint flag. If s is contracted, then it can expand and consume its flag by the next round. Otherwise, consider the case when s is expanded. If it has no children, then within the next two rounds it can contract and expand again, consuming its complaint flag; otherwise, by Lemma 26, s must have a contracted child with which it can perform a handover to become contracted in C_{i+1} and then expand and consume its complaint flag by C_{i+2}. In any case, (i) is satisfied.

Case 2: No super-root in $A(C_i)$ holds a complaint flag and not all complaint flags have been moved from follower particles to particles in layer 1. Let $p_1, p_2, \ldots, p_z$ be a sequence of particles in layer 1 such that each particle holds a complaint flag, no follower child of any particle except p_z holds a complaint flag, and no particles between the next super-root s and p_1 hold complaint flags. Then, as each p_i forwards its flag to p_{i-1} according to Definition 7.4.2, the follower child of p_z holding a flag is able to forward its flag to p_z, satisfying (ii).

Case 3: No super-root in $A(C_i)$ holds a complaint flag and all remaining complaint flags are held by particles in layer 1. By Definition 7.4.2, since no preference needs to be given to flags entering layer 1, all remaining flags will move one position closer to a super-root in each round, satisfying (iii). □

We use Lemma 30 to show first that layer 1 will be filled with particles (some possibly still expanded) in $\mathcal{O}(n)$ rounds. From that point on, in another $\mathcal{O}(n)$ rounds, one can guarantee that expanded particles in layer 1 will each contract in a handover with a follower particle, and hence all particles in layer 1 will be contracted, as we see in the following lemma:

Lemma 31. *After $\mathcal{O}(n)$ rounds, layer 1 must be filled with contracted particles.*

Proof. We first prove the following claim:

Claim 4. *After $8B_1 + 2$ rounds of $\mathcal{S}$, layer 1 must be filled with particles.*

Proof. Suppose to the contrary that after $8B_1 + 2$ rounds, layer 1 is not completely filled with particles. Then none of these rounds could have satisfied (iv) of Lemma 30, so one of $(i), (ii)$, or (iii) must be satisfied every two rounds. Case (i) can be satisfied at most B_1 times (accounting for at most $2B_1$ rounds), since a super-root expands into an unoccupied position of layer 1 each time a complaint flag is consumed. Case (iii) can also be satisfied at most B_1 times (accounting for at most $2B_1$ rounds), since once all remaining complaint flags are in layer 1, every flag must reach a super-root in B_1 moves. Thus, the remaining $8B_1 + 2 - 2B_1 - 2B_1 = 4B_2 + 2$ rounds must satisfy (ii) $2B_1 + 1$ times, implying that $2B_1 + 1$ flags reached particles in layer 1 from follower children. But each particle can hold at most one complaint flag, so at least $B_1 + 1$ flags must have been consumed and the super-roots have collectively expanded into at least $B_1 + 1$ unoccupied positions, a contradiction. □

By the claim, it will take at most $8B_1 + 2$ rounds until layer 1 is completely filled with particles (some possibly expanded). In at most another B_1 rounds, every expanded particle in layer 1 will contract in a handover with a follower particle (since $B_1 \leq n$), and hence all particles in layer 1 will be contracted after $\mathcal{O}(B_1) = \mathcal{O}(n)$ rounds. □

Once layer 1 is filled, the leader election primitive can proceed. The full description of the Universal Coating algorithm in Section 6 uses a node-based version of the leader election algorithm in [26] for this primitive. For consistency, we kept this description of the primitive here as well. However, in order to formally prove with high probability guarantees on the runtime of our universal coating algorithm, we use a Monte Carlo variant of the leader election algorithm in [26]. A description of this variant and its corresponding proofs appear in [27]. This updated algorithm elects a leader with high probability and gives the following runtime bound.

Lemma 32. *Within $\mathcal{O}(n)$ further rounds, a position of layer 1 has been elected as the leader position, w.h.p.*

Once a leader position has been elected and either no more followers exist (if $n \leq B_1$) or all positions are completely filled by contracted particles (which can be checked in an additional $\mathcal{O}(B_1)$ rounds), the particle currently occupying the leader position becomes the leader particle. Once a leader has emerged, the particles on layer 1 retire, which takes $\mathcal{O}(B_1)$ further rounds. Together, we get:

Corollary 2. *The worst-case number of rounds for $\mathcal{S}^*$ to complete layer 1 is $\mathcal{O}(n)$, w.h.p.*

7.4.4 Higher layers

We again use the dominance results we proved in Section 7.4.2 to focus on parallel schedules when proving an upper bound on the worst-case number of rounds — denoted by $Layer(i)$ — for building layer i once layer $i-1$ is complete, for $2 \leq i \leq N$. The following lemma provides a more general result which we can use for this purpose.

Lemma 33. *Consider any greedy forest–path schedule $\mathcal{S} = (A, (C_0, \ldots, C_t), L)$ with $L = v_1 v_2 \cdots v_\ell$ and any k such that $1 \leq k \leq \ell$. If every expanded parent in C_0 has at least one contracted child, then in at most $2(\ell + k)$ configurations, nodes $v_{\ell-k+1} \cdots v_\ell$ will be occupied by contracted particles.*

Proof. Let s be the super-root closest to v_ℓ, and suppose s initially occupies node v_i in C_0. Additionally, suppose there are at least k active particles in C_0 (otherwise, we do not have sufficient particles to occupy k nodes of L). Argue by induction on k, the number of nodes in L starting with v_ℓ which must be occupied by contracted particles. First suppose that $k = 1$. By Lemma 26, every expanded parent has at least one contracted child in any configuration C_j, so s is always able to either expand forward into an unoccupied node of L if it is contracted or contract as part of a handover with one of its children if it is expanded. Thus, in at most $2(\ell+k) = 2\ell+2$ configurations, s has moved forward ℓ positions, is contracted, and occupies its final position $v_{\ell-k+1} = v_\ell$.

Now suppose that $k > 1$ and that each node $v_{\ell-x+1}$, for $1 \leq x \leq k-1$, becomes occupied by a contracted particle in at most $2(\ell+k-1) = 2(\ell+k)-2$ configurations. It suffices to show that $v_{\ell-k+1}$ also becomes occupied by a contracted particle in at most two additional configurations. Let p be the particle currently occupying $v_{\ell-k+1}$ (such a particle must exist since we supposed we had sufficient particles to occupy k nodes and $\mathcal{S}$ ensures the particles follow this unique path). If p is contracted in $C_{2(\ell+k)-2}$,

then it remains contracted and occupying $v_{\ell-k+1}$, so we are done. Otherwise, if p is expanded, it has a contracted child q by Lemma 26. Particles p and q thus perform a handover in which p contracts to occupy only $v_{\ell-k+1}$ at $C_{2(\ell+k)-1}$, proving the claim. □

For convenience, we introduce some additional notation. Let t_i (resp., C_{t_i}) be the round (resp., configuration) in which layer i becomes complete.

When coating some layer i, each root particle either moves either (a) through the nodes in layer i in the set direction *dir* (CW or CCW) for layer i, or (b) through the nodes in layer $i+1$ in the opposite direction over the already retired particles in layer i until it finds an empty position in layer i. We bound the worst-case scenario for these two movements independently in order to get a an upper bound on $Layer(i)$. Let $L_i = v_1, \ldots, v_{B_i}$ be the path of nodes in layer i listed in the order that they appear from the marker position v_1 following direction *dir*, and let $\mathcal{S}_i = (A, (C_{t_{i-1}+1}, \ldots, C_{t_i}), L_i)$ be the greedy forest–path schedule where $(A, (C_{t_{i-1}+1}, \ldots, C_{t_i}))$ is a section of $\mathcal{S}^*$. By Lemma 33, it would take $\mathcal{O}(B_i)$ rounds for all (a) movements to complete; an analogous argument shows that all (b) movements complete in $\mathcal{O}(B_{i+1}) = \mathcal{O}(B_i)$ rounds. This implies the following lemma:

Lemma 34. *Starting from configuration $C_{t_{i-1}+1}$, the worst-case additional number of rounds for layer i to become complete is $\mathcal{O}(B_i)$.*

Putting it all together, for layers 2 through N:

Corollary 3. *The worst-case number of rounds for $\mathcal{S}^*$ to coat layers 2 through N is $\mathcal{O}(n)$.*

Proof. Starting from configuration C_{t_1+1}, it follows from Lemma 34 that the worst-case number of rounds for $\mathcal{S}^*$ to reach a legal coating of the object is upper bounded

by

$$\sum_{i=2}^{N} Layer(i) \leq c \sum_{i=2}^{N} B_i = \Theta(n),$$

where $c > 0$ is a constant. □

Combining Corollaries 2 and 3, we get that $\mathcal{S}^*$ requires $\mathcal{O}(n)$ rounds w.h.p. to coat any given valid object O starting from any valid initial configuration of the set of particles P. By Lemmas 27 and 28, the worst-case behavior of $\mathcal{S}^*$ is an upper bound for the runtime of our Universal Coating algorithm, so we conclude:

Theorem 13. *The total number of asynchronous rounds required for the Universal Coating algorithm to reach a legal coating configuration, starting from an arbitrary valid instance (P, O), is $\mathcal{O}(n)$ w.h.p., where n is the number of particles in the system.*

7.5 Simulation Results

In this section we present a brief simulation-based analysis of our algorithm which shows that in practice our algorithm exhibits a better than linear average competitive ratio. Since $\mathrm{OPT}(P, O)$ (as defined in Section 7.3) is difficult to compute in general, we investigate the competitiveness with the help of an appropriate lower bound for $\mathrm{OPT}(P, O)$. Recall the definitions of the distances $d(p, q)$ and $d(p, U)$ for $p, q \in V_{\mathrm{eqt}}$ and $U \subseteq V_{\mathrm{eqt}}$. Consider any valid instance (P, O). Let $\mathcal{L}$ be the set of all legal particle positions of (P, O); that is, $\mathcal{L}$ contains all sets $U \subseteq V_{\mathrm{eqt}}$ such that the positions in U constitute a coating of the object O by the particles in the system.

We compute a lower bound on $\mathrm{OPT}(P, O)$ as follows. Consider any $U \in \mathcal{L}$, and let $G(P, U)$ denote the complete bipartite graph on partitions P and U. For each edge $e = (p, u) \in P \times U$, set the cost of the edge to $w(e) = d(p, u)$. Every perfect matching in $G(P, U)$ corresponds to an assignment of the particles to positions in the coating. The maximum edge weight in a matching corresponds to the maximum distance a

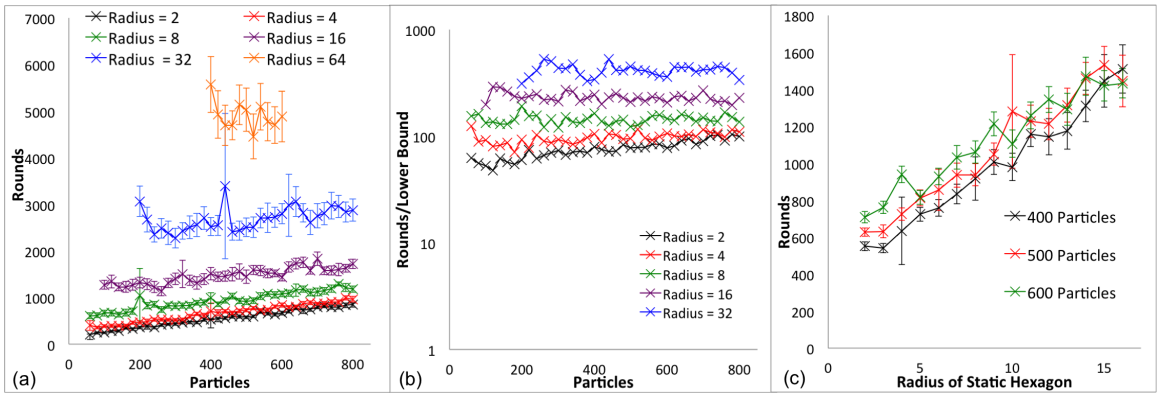

Figure 7.4: (a) shows the number of rounds varying the number of particles. (b) shows the ratio of number of rounds to the lower bound in log scale. (c) shows the number of rounds varying the static hexagon radius.

particle has to travel in order to take its place in the coating. Let $M(P, U)$ be the set of all perfect matchings in $G(P, U)$. We define the *matching dilation* of (P, O) as

$$\mathrm{MD}(P, O) = \min_{U \in \mathcal{L}} \left(\min_{M \in M(P,U)} \left(\max_{e \in M} (w(e)) \right) \right).$$

Since each particle has to move to some position in U for some $U \in \mathcal{L}$ to solve the coating problem, we have $\mathrm{OPT}(P, O) \geq \mathrm{MD}(P, O)$. The search for the matching that minimizes the maximum edge cost for a given $U \in \mathcal{L}$ can be realized efficiently by reducing it to a flow problem using edges up to a maximum cost of c and performing binary search on c to find the minimal c such that a perfect matching exists. We note that our lower bound is not tight. This is due to the fact that it only respects the distances that particles have to move but ignores the congestion that may arise, i.e., in certain instances the distances to the object might be very small, but all particles may have to traverse one "chokepoint" and thus block each other.

We implemented the Universal Coating algorithm in the amoebot simulator (see [1] for videos). For simplicity, each simulation is initialized with the object O as a regular hexagon of object particles; this is reasonable since the particles need only know where their immediate neighbors in the object's border are relative to themselves, which can

be determined independently of the shape of the border. The particle system P is initialized as idle particles attached randomly around the hexagon's perimeter. The parameters that were varied between instances are the radius of the hexagon and the number of (initially idle) particles in P. Each experimental trial randomly generates a new initial configuration of the system.

Figure 7.4(a) shows the number of rounds needed to complete the coating with respect to the hexagon object radius and the number of particles in the system. The number of rounds plotted are averages over 20 instances of a given $|P|$ with 95% confidence intervals. These results show that, in practice, the number of rounds required increases linearly with particle system size. This agrees with our expectations, since leader election depends only on the length of the object's surface while layering depends on the total number of particles. Figure 7.4(b) shows the ratio of the number of rounds to the matching dilation of the system. These results indicate that, in experiment, the average competitive ratio of our algorithm may exhibit closer to logarithmic behaviors. Figure 7.4(c) shows the number of rounds needed to complete the coating as the radius of the hexagon object is varied. The runtime of the algorithm appears to increase linearly with both the number of active particles and the size of the object being coated, and there is visibly increased runtime variability for systems with larger radii.

Chapter 8

CONCLUSIONS

In this dissertation we considered 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 investigated the feasibility of solving fundamental problems relevant for programmable matter. As a model for such self-organizing particle systems, we proposed general amoebot model which abstracts from any geometry information as well as the geometric variant of the model called geometric amoebot model. Based on the geometric model, we presented efficient local-control algorithms for leader election and line formation problems requiring only particles with constant size memory. We also discussed the limitations of solving these problems within the general amoebot model. We showed that the leader election problem in the geometric amoebot model can be decided deterministically if all the particles have a common orientatio and presented an algorithm for that. As before, particles are anonymous, have constant-size memory, have a common chirality (i.e. they are able to distinguish clockwise from counter-clockwise rotation), and utilize only local interactions. Moreover, we presented a general algorithmic framework for shape formation problems in SOPS, and showed direct applications of this framework to the problems of having the particle system self-organize to form a hexagonal or triangular shape. Our algorithms utilize only local control, require only constant-size memory particles, and are asymptotically optimal in terms of the total number of movements needed to reach the desired shape. We then developed a work-optimal *Universal Coating* algorithm for the smart paint problem on SOPS. The algorithm needed to seamlessly adapt to any valid object O, uniformly coating the object by forming mul-

tiple coating layers if necessary. We then evaluated the performance of our algorithm in terms of amount of work and total elapsed time until it terminates. In particular, we showed that the proposed algorithm terminates within a *linear number of rounds* with high probability. In addition, we presented a matching linear lower bound that holds with high probability. This lower bound is used to show a *linear lower bound on the competitive gap* between fully local coating algorithms and coating algorithms that rely on global information, which implies that our proposed algorithm is also optimal in a competitive sense. Simulation results show that the competitive ratio of our algorithm may be better than linear in practice.

One may refer to [54, 53, 7] for other work done by the Ph.D. candidate which is not directly related to this dissertation.

REFERENCES

[1] https://sops.engineering.asu.edu/simulations/.

[2] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(11):1021–1024, 1994.

[3] C. Agathangelou, C. Georgiou, and M. Mavronicolas. A distributed algorithm for gathering many fat mobile robots in the plane. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 250–259. ACM, 2013.

[4] R. Ananthakrishnan and A. Ehrlicher. The forces behind cell movement. *International Journal of Biological Sciences*, 3(5):303–317, 2007.

[5] 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.

[6] 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.

[7] S. Banerjee, A. Das, A. Mazumder, Z. Derakhshandeh, and A. Sen. On the impact of coding parameters on storage requirement of region-based fault tolerant distributed file system design. In *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pages 78–82. IEEE, 2014.

[8] 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.

[9] A. Beckers and T. Worsch. A perimeter–time CA for the queen bee problem. *Parallel Computing*, 27(5):555–569, 2001.

[10] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414(6862):430–434, 2001.

[11] A. Bhattacharyya, M. Braverman, B. Chazelle, and H. Nguyen. On the convergence of the hegselmann-krause system. *CoRR*, abs/1211.1909, 2012.

[12] L. Blázovics, K. Csorba, B. Forstner, and H. Charaf. Target tracking and surrounding with swarm robots. In *ECBS*, pages 135–141, 2012.

[13] L. Blázovics, T. Lukovszki, and B. Forstner. Target surrounding solution for swarm robots. In *EUNICE*, pages 251–262, 2012.

[14] D. Boneh, C. Dunworth, R. J. Lipton, and J. Sgall. On the computational power of DNA. *Discrete Applied Mathematics*, 71:79–94, 1996.

[15] V. Bonifaci, K. Mehlhorn, and G. Varma. Physarum can compute shortest paths. In *ACM SODA*, pages 233–240, 2012.

[16] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[17] 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.

[18] B. Chazelle. Natural algorithms. In *Proc. of ACM-SIAM SODA*, pages 422–431, 2009.

[19] H.-L. Chen, D. Doty, D. Holden, C. Thachuk, D. Woods, and C.-T. Yang. Fast algorithmic self-assembly of simple shapes using random agitation. In *DNA Computing and Molecular Programming*, pages 20–36. Springer, 2014.

[20] M. Chen, D. Xin, and D. Woods. Parallel computation using active self-assembly. In *DNA Computing and Molecular Programming*, pages 16–30. Springer, 2013.

[21] 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.

[22] G. Chirikjian. Kinematics of a metamorphic robotic system. In *Proceedings of ICRA '94*, volume 1, pages 449–455, 1994.

[23] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.

[24] R. Cohen and D. Peleg. Local spreading algorithms for autonomous robot systems. *Theoretical Computer Science*, 399(1-2):71–82, 2008.

[25] 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 the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 267–276, 2010.

[26] J. Daymude, 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 *arXiv:1503.07991.*, 2016.

[27] J. J. Daymude, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Leader election with high probability for self-organizing programmable matter. *CoRR*, abs/1701.03616, 2017.

[28] 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.

[29] 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 STACS '11*, pages 201–212, 2011.

[30] Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Brief announcement: amoebot - a new model for programmable matter. In *ACM SPAA*, pages 220–222, 2014.

[31] Z. Derakhshandeh, R. Gmyr, A. Porter, A. W. Richa, C. Scheideler, and T. Strothmann. On the runtime of universal coating for programmable matter. In *DNA22.*, 2016. It also appears in arXiv:1606.03642. Submitted to Natural Computing, Invited submission.

[32] 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 *NANOCOM*, pages 21:1–21:2, 2015.

[33] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal shape formation for programmable matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 289–299. ACM, 2016.

[34] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal coating for programmable matter. *Theoretical Computer Science, dx.doi.org/10.1016/j.tcs.2016.02.039*, 2016. It also appears as arXiv:1601.01008. 2016.

[35] Z. Derakhshandeh, R. Gmyr, T. Strothmann, R. Bazzi, A. Richa, and C. Scheideler. Brief announcement: On the feasibility of leader election and shape formation with self-organizing programmable matter. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 67–69. ACM, 2015.

[36] 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 *DNA 21.*, pages 117–132., 2015.

[37] D. Doty. Theory of algorithmic self-assembly. *Comm. of ACM*, 55(12):78–88, 2012.

[38] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. *Algorithmica*, 65(3):562–583, 2013.

[39] 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.

[40] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss. Self organizing robots based on cell structures - cebot. In *Proceedings of IROS '88*, pages 145–150, 1988.

[41] 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.

[42] A. Itai and M. 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.

[43] S. Kernbach, editor. *Handbook of Collective Robotics – Fundamentals and Challanges*. Pan Stanford Publishing, 2012.

[44] 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*, pages 159–166, 2011.

[45] G. P. Kumar and S. Berman. Statistical analysis of stochastic multi-robot boundary coverage. In *ICRA*, pages 74–81, 2014.

[46] K. Li, K. Thomas, C. Torres, L. Rossi, and C.-C. Shen. Slime mold inspired path formation protocol for wireless sensor networks. In *ANTS*, pages 299–311, 2010.

[47] N. A. Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.

[48] J. McLurkin. *Analysis and Implementation of Distributed Algorithms for Multi-Robot Systems*. PhD thesis, Massachusetts Institute of Technology, 2008.

[49] O. Michail and P. G. Spirakis. Simple and efficient local codes for distributed stable network construction. In *ACM PODC*, pages 76–85, 2014.

[50] M. J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.

[51] T. Pavlic, S. Wilson, G. Kumar, and S. Berman. An enzyme-inspired approach to stochastic allocation of robotic swarms around boundaries. In *ISRR*, pages 16–19, 2013.

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

[53] S. Shirazipourazad, Z. Derakhshandeh, and A. Sen. Analysis of on-line routing and spectrum allocation in spectrum-sliced optical networks. In *2013 IEEE International Conference on Communications (ICC)*, pages 3899–3903. IEEE, 2013.

[54] S. Shirazipourazad, C. Zhou, Z. Derakhshandeh, and A. Sen. On routing and spectrum allocation in spectrum-sliced optical networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 385–389. IEEE, 2013.

[55] T. Toffoli and N. Margolus. Programmable matter: concepts and realization. *Physica D: Nonlinear Phenomena*, 47(1):263–272, 1991.

[56] X. Trepat, M. R. Wasserman, T. E. Angelini, E. Millet, D. A. Weitz, J. P. Butler, and J. J. Fredberg. Physical forces during collective cell migration. *Nature physics*, 5(6):426–430, 2009.

[57] J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17(2):171–189, 2004.

[58] Y. Wang, P. Brown, and Y. Xia. Nanomedicine: Swarming towards the target. *Nature Materials*, 10(7):482–483, 2011.

[59] S. Wilson, T. Pavlic, G. Kumar, A. Buffin, S. C. Pratt, and S. Berman. Design of ant-inspired stochastic control policies for collective transport by robotic swarms. *Swarm Intelligence*, 8(4):303–327, 2014.

[60] 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.

[61] D. Woods. Intrinsic universality and the computational power of self-assembly. In *Machines, Computations and Universality.*, pages 16–22, 2013.

[62] D. Woods, H.-L. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *ITCS*, pages 353–354, 2013.

[63] 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.