Deterministic Leader Election in Programmable Matter

Yuval Emek
Faculty of Industrial Engineering and Management, Technion – IIT, Haifa, Israel
yemek@technion.ac.il

Shay Kutten
Faculty of Industrial Engineering and Management, Technion – IIT, Haifa, Israel
kutten@ie.technion.ac.il

Ron Lavi
Faculty of Industrial Engineering and Management, Technion – IIT, Haifa, Israel
ronlavi@ie.technion.ac.il

William K. Moses Jr.[1]
Faculty of Industrial Engineering and Management, Technion – IIT, Haifa, Israel
wkmjr3@gmail.com

Abstract

Addressing a fundamental problem in programmable matter, we present the first deterministic algorithm to elect a unique leader in a system of connected amoebots assuming only that amoebots are initially contracted. Previous algorithms either used randomization, made various assumptions (shapes with no holes, or known shared chirality), or elected several co-leaders in some cases.

Some of the building blocks we introduce in constructing the algorithm are of interest by themselves, especially the procedure we present for reaching common chirality among the amoebots. Given the leader election and the chirality agreement building block, it is known that various tasks in programmable matter can be performed or improved.

The main idea of the new algorithm is the usage of the ability of the amoebots to move, which previous leader election algorithms have not used.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Computing methodologies → Mobile agents; Theory of computation → Self-organization

Keywords and phrases programmable matter, geometric amoebot model, leader election

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.140

Category Track C: Foundations of Networks and Multi-Agent Systems: Models, Algorithms and Information Management

Related Version A full version of the paper is available at `http://arxiv.org/abs/1905.00580`.

Funding *Yuval Emek*: The work of Y. Emek was supported in part by an Israeli Science Foundation grant number 1016/17.
Shay Kutten: The work of this author was supported in part by a grant from the Bi-national Science Foundation.
Ron Lavi: This research was supported by the ISF-NSFC joint research program (grant No. 2560/17).
William K. Moses Jr.: The work of this author was supported in part by a grant from the Israeli Ministry of Science.

[1] Corresponding author.

1 Introduction

The notion of programmable matter [19], and specifically ameobots [8, 7], envisions matter as composed of tiny weak robots called “particles”. Multiple studies have addressed what these particles can achieve by cooperation, and how such weak entities can even cooperate. See e.g., coating of materials [12, 11, 4], bridge building [1], shape formation [9, 3, 10, 15, 5], and shape recovery [14]. An important primitive used often for coordinating such tasks is the election of a unique leader. Interestingly, all deterministic algorithms either elected multiple co-leaders in cases of symmetrical shapes of the matter, or relied on various assumptions on the particles, such as initially forming a specific shape (no holes), or initially having a common chirality.

1.1 Amoebot Model

Under the *amoebot model* [8, 7], each *particle* (an *amoebot*) occupies (alone) a different intersection (or *node*) of the lines of a triangular grid embedded in the plane, as seen in Figure 1.[2] The *degree* of a particle is the number of particles occupying neighboring nodes. A particle is either *contracted* (occupies one node) or *expanded* (occupies two neighboring nodes).

Each particle is *activated* infinitely often by an asynchronous scheduler to act. One *asynchronous round* is completed when each particle is activated at least once. The activation of a particle is atomic, i.e., it is completed before the next particle is activated. Each activation consists of 3 stages: (i) P reads the memories of adjacent particles, (ii) P performs some local computation and may update its own memory and/or the memories of its neighboring particles (sometimes, this is called “sending messages”), and (iii) P may move by either expanding or contracting.[3] Specifically, an expanded particle can contract to either one of the two nodes it occupies. While contracting out of node b, the particle can *pull* a contracted particle Q that occupies a node c that neighbors b; then Q becomes expanded and occupies both c and b. The expansion of P from node b into a neighboring node c is possible if P is contracted and c is not occupied by another contracted particle. After the expansion, P occupies both b (termed P's *tail*) and c (termed P's *head*). Suppose that P's move is an expansion into c, already occupied by a different particle Q who is expanded (Q occupies also a different node e). We then say that Q is *pushed*, Q becomes contracted and occupies only node e. For the sake of convenience, we may sometimes view the (occupied) nodes as the entities taking actions.

Each particle has constant size memory. In the *leader election (LE)* problem, each particle has one of three *LE statuses*: **C** (candidate, the initial state), **L** (leader), and **U** (unelected). and will permanently change its status to either **U** or **L** such that exactly one particle has status **L** by the end of the algorithm.

The particles are classified according to their *chirality* as *clockwise (CW) particles* or *counter-clockwise (CCW)* so that a CW (resp., CCW) particle numbers the *ports* corresponding to the 6 incident edges in (each of) the node(s) it occupies from 0 to 5 in increasing CW (resp., CCW) order; the edge from which this numbering starts is chosen arbitrarily (refer to Figure 2 for an illustration). We assume that the particle chirality classification is

[2] Because of space constraints, all figures are found in the full version.
[3] Note that P allocates memory for each of its ports and that is the memory that can be modified by adjacent particles. In other words, when P receives a message, it knows through which port the message was sent and by extension which particle sent it.

determined by a malicious adversary and that initially, a particle does not know whether its chirality (or a chirality of any other node) is CW or CCW.

The *configuration* of the particle system at any given time is comprised of the location and state of each particle; it is *contracted* if all particles are contracted. We follow the common practice (see [6]) and assume that the particles are, initially, in a contracted configuration. The algorithm terminates in a contracted configuration.

Define a graph $G(t)$, called the *shape*, induced on the grid by the nodes occupied by particles at time t and an edge connects two graph nodes if they represent two neighboring grid nodes. Following the common practice in the amoebot model literature (see [8]), it is required that the shape is connected at all times. Since the shape is a (finite) planar graph associated with a planar embedding, it partitions the plane into *faces* (see [16]), where exactly one of them is the *outer* face. The occupied nodes adjacent to the outer face are said to form the *outer boundary* of the shape. An inner face that includes at least one unoccupied node is called a *hole* in the shape (refer to Figure 3 for an example). The occupied nodes adjacent to a hole are said to form an *inner boundary.* The *length* of a boundary B, denoted L_B, is the number of nodes on B; those are *boundary nodes*, occupied by *boundary particles.* Define $L_{\max} = \max_B L_B$.

1.2 Related Work

Randomized algorithms, assuming common chirality in the initial configuration, are given in [13, 6]. A deterministic algorithm was presented in [15] to both elect a leader and obtain common chirality for the natural special case that the shape did not contain holes; multiple leaders could be elected in some cases (a constant number). They then used the leader(s) to coordinate shape transformation. The no-holes assumption is replaced in [2] by an assumption that the particles start with a common chirality. In a brief announcement, they outlined an interesting algorithm that still may end with several leaders in cases of high symmetry. The current paper adapts and uses a large part of the algorithm of [2] as a procedure. In [17], both common chirality and no-holes are assumed to elect a unique leader. That algorithm also assigns, to each particle, an identifier that is unique within a radius of k. Moreover, beside triangular grids, their algorithm can work also on the square and king grids.[4]

The type of asynchronous scheduler used affects the leader election results. Typically in the literature [13, 6, 17], the scheduler provides conflict resolution mechanisms for movement and communication such that particle activations can be analyzed sequentially, i.e., the activation of each particle is atomic. As the current paper demonstrates, it is *not* impossible to elect a leader deterministically (unless, perhaps, in models when a scheduler is allowed to schedule such particles simultaneously [2, 15]).

1.3 Technical Challenges and Ideas

Multiple ideas are combined here in order to address different cases. Consider, for example, a polygon with a hole. One approach in previous algorithms, assuming no holes, was to remove (from being candidates) boundary nodes repeatedly until only one remains. In the case of one hole (addressed by one of our subroutines), the present algorithm utilizes the ability of particles to move. Intuitively, they may move (eventually) to the center of the polygon, and the particle reaching the center first is the elected one. (Thanks to the sequential scheduler, only one can reach a certain node first).

[4] It is possible to adapt the current paper's leader election algorithm to run on king grids, but this would require some work, while it follows in a natural way for the algorithm of [17].

This, of course, requires our algorithm to perform various maneuvers, to identify the center and to make sure no additional holes remain. In particular, particles have to identify the outer boundary, move outward in order to gain a symmetric shape, and then move inward together so no additional holes are formed. Since multiple polygons may be moving at the same time, two polygons may "collide" and not manage to finish the maneuver. There, we use the idea of reset, to restart the algorithm for the new shape. We managed to upper bound the number of such resets.

Because of the existence of bridge (and semi-bridge particles) (to be defined in the next section, but intuitively particles whose removal disconnects the shape), solving for a single simple polygon is not enough. For example, consider the case that the shape is a long line (possibly connecting simple polygons). Here, we use the fact that there exists only one outer face (borrowing its detection from the algorithm of [2], with some necessary adaptations). The partial leaders of the simple polygons cooperate to define a tree that spans the simple polygons. Final leader election is then performed over the tree.

The assumption of common chirality is used throughout the paper. To remove this assumption and have particles agree on chirality, we use again the detection of the outer face. The particles on the outer boundary agree on chirality (this turned out to be easier for us than agreeing on a leader among them, using the local symmetry breaking provided by the scheduler). Then, the outer boundary particles coordinate and propagate this shared chirality to the other particles within the shape.

1.4 Our Contributions and Paper Organization

The current paper presents the first deterministic protocol that elects exactly one leader on any contracted configuration, even without assuming common chirality. For a comparison of this result to known ones, please see Table 1.

The building blocks may be of interest by themselves. The assumption of common chirality is removed last, in Section 5. Other building blocks are: maximal independent set (MIS) protocol, boundary detection, leader election on a convex polygon without sharp vertices, and leader election on a spanning tree. They are are given in Section 3. Additional definitions required to understand the paper are present in Section 2. The main protocol, `Leader` – `Election` – `By` – `Moving` (before removing the common chirality assumption), is presented in Section 4 (together with some other small components, such as broadcast with termination detection, and reset).

Table 1 Table comparing the result on leader election to those of previous papers. "No holes" refers to whether the algorithm requires the graph to have no holes initially or not. "Multiple leaders" refers to whether the leader election algorithm may output multiple leaders in certain cases or always outputs a unique leader. The length of the largest boundary in the initial configuration is $L_{\max}$. The length of the outer boundary in the initial configuration is L. The number of particles in the configuration is denoted by n. The terms r and $mtree$ are unique to paper [17].

Paper	Common chirality	Randomness	No holes	Multiple leaders	Running time
[13]	Yes	Yes	No	No	$O(L_{\max})$ rounds on expectation
[6]	Yes	Yes	No	No	$O(L)$ rounds with high probability
[2]	Yes	No	No	Yes	Not analyzed in paper
[15]	No	No	Yes	Yes	$O(n)$ rounds
[17]	Yes	No	Yes	No	$2(r + mtree + 1)$ rounds
Current Paper	No	No	No	No	$O(Ln^2)$ rounds

2 Preliminaries: Shape and Boundaries

We need quite a few definitions related to the shape. A *local boundary* of a particle is an interval $i, i+1, \ldots, i+j \mod 6$ of its ports that lead to unoccupied grid nodes. Note that a contracted particle may have up to three local boundaries, each a part of some boundary of the shape. However, all three may be parts of the same boundary of the shape. Henceforth, we use only the term "boundary" even for local boundaries, when the context makes the usage clear. A *bridge particle* is a *contracted* boundary particle occupying a node b lying on i local boundaries (note that $1 \leq i \leq 3$), each of which is a part of the outer boundary, and having i occupied adjacent nodes in the grid. An example of bridge particles and semi-bridge particles (see next paragraph) with bridge edges is illustrated in Figure 4.

A *semi-bridge particle* is a *contracted* boundary particle occupying a node b lying on 2 outer boundaries, and having 3 or 4 occupied adjacent nodes in the grid. If b is occupied by a bridge or semi-bridge particle, c is an adjacent occupied node, and both sides of the edge (b, c) are on the outer boundary, then edge (b, c) is called a *bridge edge.*[5]

For a boundary node b occupied by particle P with chirality C and lying on boundary B, define b's *predecessor* node a and *successor* node c w.r.t. B and C as the previous occupied node and the next occupied node along B according to C, respectively (refer to Figure 5 for an illustration).[6] Note that node b admits such predecessor a and successor c for each boundary b lies on.

The *boundary count* of b w.r.t. B and C measures the deviation of the line segment formed by b and its successor from the line segment formed by b's predecessor and b w.r.t. B taking C into account. More formally, the boundary count of b w.r.t. B is a function of C and the angle $\angle abc$ that takes on one of the values $-1, 0, 1, 2$, or 3 (as illustrated in Figure 6).[7] Let i be the unique integer that satisfies $\angle abc = 180^\circ - i * 60^\circ$. Let x and y be the port numbers of b corresponding to edges (b, a) and (b, c), respectively. If $(x - y) \mod 6 = 4$, then the boundary count of b w.r.t. B is $-i$, else it is i. When the boundary referred to is clear from context, it is not mentioned when giving the boundary count for a node.

Consider an occupied node b on boundary B with boundary count w. The following definitions for b are all w.r.t. B. Node b is called a *vertex* when $w = -1, 1, 2$, or 3.[8] When $w = 2$, b is a *sharp* vertex. Vertex b is *concave* when $w = -1$ and *convex* when $w = 1$ or 2. A shape whose outer boundary vertices are all convex w.r.t. the outer boundary is a *convex polygon.* A shape consists of *two (or more) simple convex polygons sharing the same contracted semi-bridge particle(s) P (, Q, R, etc.)* when (i) P (, Q, R, etc.) has no adjacent bridge edges, (ii) the shape is disconnected by removing P (, Q, R, etc.), and (iii) all vertices other than those occupied by P (, Q, R, etc.) are convex vertices. Notice that the definition of convex polygon relates only to its outer boundary nodes. Specifically, no assumptions are made on the presence of holes within the shape.

[5] Note that a semi-bridge particle may have 3 adjacent occupied nodes and lie on 2 outer boundaries and 1 inner boundary. In this case, the particle still has 1 bridge edge.

[6] Throughout, we use w.r.t. to abbreviate "with respect to".

[7] Note that it is not possible to have a node with boundary count -2 or -3 w.r.t. some boundary. Also note that the boundary count and its application to calculating the count of a segment, defined and used in the full version, is similar to how [2] uses vertex labeling in deciding the count of a segment in their paper. The actual measurement of the boundary count is similar to how [13] measures the angles between the direction a token enters and exits an agent.

[8] Notice that the angle bisector of a vertex with boundary count 1 or -1 overlaps with a line of the triangular grid.

3 Building Blocks

Let us now present the four building blocks in brief, except Subsection 3.3 that is more detailed. The full version contains all missing details. The description uses some additional definitions. Each boundary particle P maintains a binary flag *seg_head* in each boundary node b that P occupies w.r.t. each boundary that b lies on. When P occupies a boundary node and has $seg\_head = true$ for that boundary, we say P is a *seg-head* for that boundary. Consider two seg-heads P_1 and P_2 on boundary B, occupying nodes b_1 and b_2 with predecessor nodes c_1 and c_2 and successor nodes d_1 and d_2 w.r.t. B, respectively. P_2 is the *previous (resp., next) seg-head* before (resp., after) P_1 iff the particles in successor (resp., predecessor) nodes from b_2 to b_1 w.r.t. B (excluding b_1 and b_2) have *seg_head* set to $false$.

Let P_2 be the next seg-head after P_1 w.r.t. B. P_1's *segment* is the sequence of successor nodes from b_1 to c_2 with *head* b_1 and *tail* c_2. It is said that P_1's segment is *before* P_2's segment or P_2's segment is *after* P_1's segment w.r.t. B. For the sake of convenience, when referring to a procedure/action initiated by the head of a segment involving the participation of the particles in that segment, we just say that a segment runs the procedure/performs the action. It is important to note that a particle P may participate in multiple segments simultaneously (one per boundary P lies on). The algorithm needs to be careful to prevent contradicting actions of such segments (for example, preventing one segment from expanding P into one node while another segment is trying to expand P into a different node).

3.1 MIS Selection

To perform Procedure `MIS − Selection`, P joins the MIS iff no neighbor of P has yet joined the MIS. The following trivial observation breaks with impossibility results in other models when the scheduler is not asynchronous.[9]

▶ **Observation 1.** *When run by particles, procedure* `MIS − Selection` *deterministically computes an MIS in one round.*

3.2 Boundary Detection

`Boundary − Detection` is a parameterized procedure run by boundary nodes with common chirality to tell each such node b, for each boundary B that b lies on, whether B is an inner or outer boundary. This procedure is a modification of the first phase of the algorithm presented in [2], specifically adapting their subroutine `StretchExpansion` to handle (1) inner boundaries and (2) an edge case that may not be needed (and is not addressed) in [2] but is needed here (see Figure 7).[10] These adaptations result in subroutines `Inner − Stretch − Expansion` and `Outer − Stretch − Expansion`, respectively. Due to space constraints, the entire modified boundary detection procedure and proof of the theorem are presented in the full version.

▶ **Theorem 1.** *When executed by contracted boundary particles, procedure* `Boundary− Detection` *terminates in $O(L^2_{\max})$ rounds resulting in each boundary node b knowing, for each boundary B it is on, whether B is an inner or outer boundary. If b has $seg\_head = true$ w.r.t. boundary B, then b knows how many nodes k, $k \in \{1, 2, 3, 6\}$, are also segment heads w.r.t. B.*

[9] In particular, this procedure selects a leader in a ring of 3 particles.
[10] Recall that [2] is a brief announcement, so this edge case may be handled in the full version of their paper.

3.3 Leader Election on a Convex Polygon without Sharp Vertices

Procedure `Convex – Polygon – Leader – Election` relies on 3 subroutines, described below. Note that the outer boundary nodes of a convex polygon without sharp vertices form a hexagon in the grid. Let b be a vertex, occupied by particle P, with successor node d w.r.t. the outer boundary. Define *P's side* as the side of the hexagon containing b and d.

Let LSLS stand for largest same length sides and SSLS stand for smallest same length sides. Every possible hexagon is isomorphic to one of the following four. See Figures 8-13.

1. Category 1: The hexagon has exactly either 1 LSLS or 1 SSLS.
2. Category 2: The hexagon has either 2 LSLS and 4 SSLS, 4 LSLS and 2 SSLS, or 2 LSLS, 2 SSLS, and 2 other same length sides.
3. Category 3: The hexagon has exactly 3 LSLS and 3 SSLS.
4. Category 4: The hexagon has 6 sides of the same length.

Let $P_0, P_1, \dots P_{k-1}$ be the seg-heads on the outer boundary such that $P_{(i+1) \mod k}$ is the next seg-head after P_i. Subroutine `Compare – Length(x)` is initiated by a seg-head P_i to compare the length of P_i's segment with that of $P_{(i+x) \mod k}$'s segment. The procedure simulates the way a Turing machine would perform a similar task, where the segments would be segments of the machine's tape. Since this is a known method, the details are omitted (refer to [15] for an example). The proof of the following lemma can be found in the full version.

▶ **Lemma 2.** *Let x be a constant and assume that there are L nodes on the outer boundary. If the nodes from P_i's segment's head to $P_{(i+x) \mod k}$'s segment's tail run* `Compare – Length(x)`*, then the subroutine terminates in $O(L^2)$ rounds, resulting in P_i knowing the size comparison between the two segments.*

Consider two parallel lines M_1 and M_2 of the grid. The *mid-line(s)* between M_1 and M_2 is the line(s) parallel to both M_1 and M_2 which is either equidistant from both M_1 and M_2 or not closer to one of the lines by more than a unit distance. Consider a category 2 hexagon where opposite outer boundary vertices b_1 and b_2, occupied by particles P_1 and P_2 respectively, have $seg\_head = true$ and the remaining nodes have $seg\_head = false$. There exist either 1 or 2 mid-lines between P_1's side and P_2's side. Let c_1 and c_2, occupied by particles Q_1 and Q_2 respectively, be nodes on P_1 and P_2's segments respectively lying on the mid-line (or on the closer mid-line to the head of the segment in the case of 2 mid-lines). The outer boundary particles run subroutine `Mid – Line` to find c_1 and c_2 and subsequently Q_1 and Q_2 set $seg\_head = true$ and P_1 and P_2 set $seg\_head = false$. See Figures 10 and 11 for examples. A more detailed description of the subroutine is found in the full version. The following observation captures the running time of `Mid – Line`.

▶ **Observation 2.** *Let there be L outer boundary nodes on a category 2 hexagon with opposite vertices b_1 and b_2, occupied by particles P_1 and P_2 respectively, with $seg\_head = true$ and remaining nodes with $seg\_head = false$. Subroutine* `Mid – Line`*, run by the L nodes, terminates in $O(L^2)$ rounds, such that nodes c_1 and c_2, which are the closest nodes in P_1 and P_2's segments lying on mid-lines between P_1's side and P_2's side respectively, now have $seg\_head = true$ and b_1 and b_2 have $seg\_head = false$.*

Intuitively, when the segment heads are on the mid-line as promised by Observation 2, if they move towards the center, they can get next to each other and elect one of them as a leader. The following subroutine `Snake – Movement(D, x)` (described in more detail in the full version), is used for election in hexagons of several types. Consider a path p of w nodes occupied by contracted particles with head node b occupied by particle P and tail node c. P has $seg\_head = true$ w.r.t. the outer boundary and the remaining particles have $seg\_head = false$. See Figure 14 for an example. Particles in p (termed *snake* p) expand, so p becomes longer and its head P moves in direction D for distance $x \leq w$ (without breaking connectivity and while keeping the tail node of p fixed at c). P is the one expanding first, and the particles perform a sequence of expansions and contractions until reaching the desired total length of p. Note that x may not be a constant. Hence, this value is represented distributively on the particles of p by them simulating a tape of a Turing machine. (The value of x is also input to the subroutine and used for the computing in the same manner).

Note that only a segment on the outer boundary can perform this procedure, hence, snake p will not belong to two different segments giving it contradictory instructions to move. One thing that may happen is that the head of snake p reaches a particle Q not in snake p. If Q belongs to another snake p' then p stops. Otherwise, p continues moving in direction D simply by annexing Q who now becomes the head of the snake (that we still call snake p). The proof sketch of the following lemma can be found in the full version.

▶ **Lemma 3.** *Assume that L contracted particles of a snake run* $\mathtt{Snake-Movement(D,x)}$, *where $x \leq L$. Then, the subroutine terminates in $O(x^2)$ rounds without breaking connectivity. On termination, either the snake head reached a node at distance x away from the head of the snake in direction D, or the next particle in direction D belongs to another snake.*

Now, procedure $\mathtt{Convex-Polygon-Leader-Election}$ is described. Note that illustrations expanding on the description are found in the full version. Initially, the 6 particles that occupy vertices on the outer boundary set $seg\_head = true$ while the remaining particles in the polygon set $seg\_head = false$. Each of these 6 particles initiates $\mathtt{Compare-Length(x)}$ for $1 \leq x \leq 6$, sends messages to the remaining 5 particles with the results of these comparisons, and determines which category hexagon it lies on.[11] The procedure follows one of the following four cases:

1. *Category 1 hexagon:* This case is trivial - the particle at the head of the smallest or largest side becomes the leader. See Figures 8 and 9.
2. *Category 2 hexagon:* If there are exactly 2 LSLS, then those sides' polygon vertices keep $seg\_head = true$ and the remaining vertices set $seg\_head = false$. Else there are 2 SSLS whose vertices keep $seg\_head = true$ while others set $seg\_head = false$. Call particles occupying vertices with $seg\_head = true$, P_1 and P_2, and denote the direction from the successor of P_1 to P_1 as D_1 (similarly denote D_2). Now, P_1 and P_2 initiate $\mathtt{Mid-Line}$ resulting in two new particles Q_1 and Q_2 setting $seg\_head = true$ and P_1 and P_2 setting $seg\_head = false$ (refer to Figures 10 and 11 for examples).
   The resulting segments of Q_1 and Q_2 form snakes p_1 and p_2 with lengths w_1 and w_2 respectively that run $\mathtt{Snake-Movement(D_1,w_1)}$ and $\mathtt{Snake-Movement(D_2,w_2)}$ in directions D_1 and D_2, respectively. In addition to the usual termination conditions when running $\mathtt{Snake-Movement(D_1,w_1)}$ and $\mathtt{Snake-Movement(D_2,w_2)}$, the subroutines also terminate when the head of p_1 is adjacent to that of p_2. Then, the two heads run $\mathtt{MIS-Selection}$ and the particle that joins the MIS becomes the leader.
3. *Category 3 hexagon:* Let P_1, P_2, and P_3 occupy vertices b_1, b_2, and b_3 such that P_1, P_2, and P_3's sides are the 3 largest sides. The remaining particles set $seg\_head = false$. See Figure 12. Let D_1, D_2, and D_3 be the directions along the angle bisectors of b_1, b_2, and b_3 respectively toward the center of the hexagon. The two phase procedure followed by P_1's segment is now described. (P_2's and P_3's segments act similarly).

[11] With this information, a particle can compute, using a constant amount of space, the total order on the lengths of sides of the hexagon. Combined with information of which sides are equal in length, a particle can determine both the category of the hexagon it lies on and the type of its own side.

Notice that P_1's segment encompasses 1 SSLS and 1 LSLS with lengths x and y respectively. In phase 1 (simulating a Turing machine), the values of $f = \lfloor (y-x)/3 \rfloor$, $g = x + f$, and $q = (y-x) \mod 3$ are computed and stored in P_1's segment. If $q = 2$, g is incremented by 1. Now P_1 sends a message to the particle Q_1 located f nodes from the head of the segment, telling Q_1 to set $seg\_head = true$ and store D_1, f, g, and q. P_1 subsequently sets $seg\_head = false$. Q_1 is now the head of a segment. Similarly, some Q_2, Q_3 replace P_2, P_3 as heads of their segments. Now Q_1 sends a message along the outer boundary to Q_2 and Q_3 indicating that the first phase is over. Once Q_1 receives similar messages from Q_2 and Q_3, the second phase begins.
In phase two, Q_1's segment runs $\mathtt{Snake-Movement(D_1, g)}$. If $q = 0$, all three snakes move towards the same final node b. Let p be the snake such that its head particle R is the first to occupy b. R waits until the remaining two snakes reach it and then becomes the leader. If $q \neq 0$, the final nodes occupied by the heads of the three snakes form a triangle. Let R be a head of the snake that occupies one of the triangle's nodes. R waits until the other two triangle's nodes are occupied and then runs $\mathtt{MIS-Selection}$. The particle chosen to be in the MIS becomes the leader.

4. *Category 4 hexagon:* All vertices have $seg\_head = true$ (e.g., Figure 13). The procedure here is a simplified version of the case of Category 3 hexagon. See the full version.

▶ **Theorem 4.** *Procedure* $\mathtt{Convex-Polygon-Leader-Election}$ *run by contracted particles of a convex polygon without sharp edges results in exactly one leader being elected deterministically in $O(L^2)$ rounds, where L is the number of particles on the outer boundary.*

Proof Sketch. The readers can convince themselves that all types of hexagons have been accounted for in the four hexagon categories. Let us prove correctness for each category separately. The case of category 1 is trivial.

In a category 2 hexagon, Observation 2 guarantees that particles are chosen such that they lie on the same mid-line or adjacent mid-lines. The distance needed to be traveled by each segment until both heads are adjacent is $\leq L/2$. Since the segments divide the nodes of the outer boundary equally, each segment has enough contracted particles such that it is possible to traverse this distance by expanding every particle in the segment. Moreover, no two snakes can block each other before reaching that distance. $\mathtt{MIS-Selection}$ is guaranteed to choose exactly one leader due to Observation 1.

For category 3, inscribe the hexagon in an equilateral triangle with vertices A, B, and D, centroid C, and F trisecting $\overline{AB}$, as seen in Figure 15. Observe that each side of the triangle is of length $2x + y$ and $|\overline{AF}| = |\overline{FC}|$. When $(y-x) \mod 3 = 0$, $\overline{FC}$ coincides with a grid line and all segments move towards C using `Snake-Movement()`. However, if $(y-x) \mod 3 \neq 0$, the segments move to nodes that form a triangle around the centroid, in which case, $\mathtt{MIS-Selection}$ is run and Observation 1 guarantees a leader is selected. Note that no two snakes can block each other before reaching the meeting point.

The proof for category 4 is a simplified version of the proof for Category 3. Thus for all four types of hexagons, a leader is chosen. The running time is analyzed in the full version. ◀

3.4 Leader Election on a Spanning Tree

Procedure $\mathtt{Spanning-Tree-Leader-Election}$ deterministically elects a unique leader when participating particles form a spanning tree and have common chirality. Note that this can also be performed by the algorithms of [15, 17]. We defer the description to the full version and give the following theorem without a proof.

▶ **Theorem 5.** *Procedure* `Spanning – Tree – Leader – Election` *run by particles forming a spanning tree of diameter x results in exactly one leader being elected deterministically in $O(x)$ rounds.*

4 Leader Election

An overview of deterministic algorithm `Leader – Election – By – Moving` for electing a unique leader is now given, assuming common chirality (an assumption removed later). Additional details and proofs appear in the full version.

The initial contracted configuration of n particles forms a connected shape $G(0)$ at the beginning of round 0 with all particles having status **C**. $H(t), K(t), F_1(t)$, and $F_2(t)$ are virtual graphs at the beginning of round t that are maintained by the particles distributively and are initially empty.[12] Note that the round number is subsequently dropped, as it is apparent from context.

The algorithm has six phases. Graph H is used throughout the algorithm for various purposes depending on the phase of the algorithm. Graph K is a subgraph of G that holds a spanning tree of all particles and is important for phase 6 of the algorithm. Graph F_1 is a forest of trees of all particles used throughout the algorithm. Graph F_2 is a forest of trees of a subset of the particles used only in phase 5 of the algorithm.

Each particle P maintains a phase counter in $[1 \dots 6]$ and appends its value to each message sent. If P receives a message from another particle Q in a different phase, P does not process Q's message until P is in the same phase as Q.[13]

1. *Initialization:* At the end of this phase, every particle is contracted and each boundary particle has identified the type (inner/outer) of each of its boundaries. Furthermore, graph H consists of a set of simple convex polygons, where two simple polygons may share the same semi-bridge particle.
   Each boundary particle runs `Boundary – Detection` for each boundary B it lies on to determine whether B is an inner or outer boundary. Once `Boundary – Detection` terminates, all particles not on the outer boundary set $seg\_head = false$. Thus, there are k, $k \in \{1, 2, 3, 6\}$, particles with $seg\_head = true$ located on the outer boundary. Call these seg-heads $P_1, P_2, \dots, P_k$. If $k = 1$, change P_1's status to **L** and broadcast (by simple flooding [18]) a $final\_terminate$ message to other particles to terminate the algorithm and change their statuses to **U**.
   Each particle that is not a bridge or semi-bridge adds itself and its edges to adjacent nodes to H. Otherwise, semi-bridge particles add themselves and their non-bridge edges to H. Note that all particles are contracted at the end of this phase.
2. *Spanning forest formation:* Each outer boundary node a becomes the root of a tree T and uses the standard broadcast-&-echo method [18] to recruit nodes to its tree. Each node b joins exactly one tree T. Termination detection of the phase is coordinated by seg-heads P_1 to P_k, after all the broadcast-&-echoes terminate. Thus a spanning forest F_1 of trees is formed with outer boundary particles as roots of the trees. Furthermore, each node knows its parent and children in the tree. Note that all particles remain contracted during the phase.

[12] For each graph, each particle maintains locally its own edges in the graph and whether it is in the graph or not. Each particle allots a constant amount of memory for each of the graphs G, H, K, F_1, F_2 and updates them as necessary when activated.

[13] It is trivial to return to a contracted configuration from the configuration the algorithm terminates in, so this "7th phase" is not described. Informally, it consists of particles that performed `Snake – Movement()` as part of `Convex – Polygon – Leader – Election` during phase 5 reversing their movements.

3. *Convexification:* The subgraph H, induced by removing bridge particles from the shape, is a collection of polygons. Each outer boundary particle P w.r.t. H that is a concave vertex and not a semi-bridge particle expands towards the outer boundary along P's angle bisector while coordinating the pulling of P's tree with it. P occupying node b and moving to node c completes one step of convexification when it has moved to node c and all particles in the tree rooted at P in F_1 are back in a contracted state. Convexification is performed repeatedly by particles until no more steps of convexification are possible (refer to Figure 16 for an example).

   At the same time, each seg-head P_i $(1 \leq i \leq k)$ continuously checks its segment for any concave vertices in H. If none are found, the k seg-heads coordinate to terminate this phase. All particles previously in H update their edges in H to reflect current connections to other particles. Bridge and semi-bridge particles add themselves and their bridge edges to virtual graph K. Note that all particles are contracted at the end of this phase.

   The phase as described so far may be stopped before convexification completes if two types of situations arise. *Type 1*: during the movement outward, an outer boundary particle that moved in some direction D to node b in one step of convexification finds out that the node adjacent to b in direction D is occupied. *Type 2*: a semi-bridge particle, bridge particle, or outer boundary particle stops being one. Both types reflect a change in the particles occupying the outer boundary, possibly resulting in particles previously with $seg\_head = true$ no longer lying on the outer boundary. The algorithm then resets to phase 1, as described in the full version (the reset procedure also makes sure that all the particles are reset to a contracted state).

4. *De-sharpification:* In this phase, certain particles remove themselves from H recursively until only convex polygons and two-node lines remain in H. Consider a particle P in H. If P is not a semi-bridge particle and is a sharp vertex w.r.t. the outer boundary in H, then P removes itself from H. If P is a semi-bridge particle and its occupied adjacent nodes are located at ports $x, x+1, x+3$, and $x+4 \pmod 6$ for some positive integer value of x, then P removes itself from H.

   At the same time, each seg-head P_i $(1 \leq i \leq k)$ checks its segment continuously for any sharp vertices in H. If none are found, P_i coordinates with the other $k-1$ seg-heads to terminate the phase. The induced subgraph H at the end of the phase is a set containing just two types of polygons: (a) lines consisting of 2 nodes as well as (2) convex polygons without sharp vertices. Note that all the particles remain contracted at the end of this phase.

5. *Leader election on individual polygons and spanning tree formation:* This phase consists of two stages. In stage one, each convex polygon and each line in H elects a unique polygon leader using $\mathtt{Convex}-\mathtt{Polygon}-\mathtt{Leader}-\mathtt{Election}$ and $\mathtt{MIS}-\mathtt{Selection}$, respectively. In stage two, each particle P chosen as a polygon leader in stage one, acts as a root and forms a tree that spans its connected component of $G \setminus K$. The nodes in K that are reachable from P over $G \setminus K$ are leaves of P's tree. Call this forest of polygon leaders rooted trees F_2.

   The termination condition is somewhat long to describe; see the full version for details. Very informally - the polygons are connected by semi-bridge particles. Hence, when a polygon leader finished constructing its tree over the polygon, the semi-bridge particle(s) is notified. The seg-head particles P_i $(1 \leq i \leq k)$ check continuously the semi-bridge particles to know when the construction of F_2 is done.

   At the end of this phase, K is updated to contain all particles in graph G with edges restricted to bridge edges and edges of F_2. It is shown later in a lemma that K now forms a tree, spanning all the candidates (status **C** particles).

6. *Leader election on a spanning tree:* Each particle participates in `Spanning − Tree − Leader − Election` on the graph K. Once a particle P changes its status to **L**, P broadcasts a $final\_terminate$ message by flooding along K. This results in one particle, the leader, having status **L** and the remaining particles having status **U** when the algorithm terminates.

The following lemmas apply to the algorithm, with proofs deferred to the full version.

▶ **Lemma 6.** *Phase 1 terminates in $O(L_m^2)$ rounds, where L_m is the length of the largest boundary of the shape, resulting in each boundary particle knowing what type each of its boundaries is and k, $k \in \{1, 2, 3, 6\}$, particles, $P_1, P_2, \ldots, P_k$, lying on the outer boundary with seg_head = true. Furthermore, at the end of the phase, H consists of a set of simple convex polygons, where two simple polygons may share the same semi-bridge particle. If $k = 1$, the algorithm terminates with one particle as leader in an additional $O(n)$ rounds.*

▶ **Lemma 7.** *Phase 2 terminates in $O(n)$ rounds, resulting in a disjoint forest of trees F_1 covering every particle.*

▶ **Lemma 8.** *Phase 3 terminates in $O(Ln)$ rounds, resulting in either a reset or a graph H containing a set of simple convex polygons, where two simple polygons may share the same semi-bridge particle.*

▶ **Lemma 9.** *There can be at most L resets occurring in phase 3, where L is the length of the outer boundary of the original shape.*

▶ **Lemma 10.** *Phase 4 takes $O(n)$ rounds to complete, resulting in H containing only a set of lines consisting of 2 nodes and convex polygons without sharp vertices.*

▶ **Lemma 11.** *Phase 5 terminates in $O(L^2+n)$ rounds resulting in K containing all particles and forming a spanning tree.*

▶ **Lemma 12.** *Phase 6 terminates in $O(n)$ rounds resulting in a unique leader with status **L** being chosen and all other nodes having status **U**.*

Combining the above lemmas together, we get the desired results.

▶ **Theorem 13.** *Algorithm* `Leader − Election − By − Moving`*, run by n particles in a contracted configuration, elects a unique leader deterministically and terminates in $O(Ln^2)$ rounds, where L is the number of particles on the outer boundary of the original shape.*

Proof Sketch. From Lemmas 6, 7, and 8, the combined running time of one iteration of phases 1 to 3 is $O(n^2)$ rounds since $L_m = O(n)$. There can be at most $O(L)$ iterations of phases 1 to 3, by Lemma 9. Adding in the running times of phases 4 to 6 from Lemmas 10, 11, and 12, it is clear that the total running time of the algorithm is $O(Ln^2)$ rounds.
The correctness directly follows from Lemma 12. ◀

5 Chirality Agreement

In this section, procedure `Chirality − Agreement` is described. Consider n contracted particles forming a connected shape with the length of maximum boundary being $L_{\max}$. The particles run `Chirality − Agreement` and terminate in $O(L_{\max}^2 + n)$ rounds, resulting in all particles agreeing on the same chirality and forming the original shape.

Informally, after the boundary particles identify their boundaries, they first agree on chirality separately for each boundary they lie on. This is easier than leader election given the local symmetry breaking built into the model (atomicity of the scheduler). This is enough to allow the particles to identify the outer boundary similarly to the way it was done for the leader election. Finally, the chirality agreed upon for the outer boundary becomes the chirality of everyone. A detailed explanation of the procedure along with the proof of the following theorem can be found in the full version.

▶ **Theorem 14.** *Procedure* `Chirality − Agreement`*, run by n contracted particles forming a connected shape, terminates in $O(L_{\max}^2)$ rounds, where $L_{\max}$ is the length of the largest boundary in the shape, resulting in all particles having common chirality and retaining the original shape.*

6 Conclusion and Future Work

The results of this paper leave several lines of research open. First, the algorithms here require the particles to move for leader election and for chirality agreement. Is it possible to solve either problem deterministically in the given setting without requiring particles to move? Second, can one reduce the running time or provide a matching lower bound?

References

1. Marta Andrés Arroyo, Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. A stochastic approach to shortcut bridging in programmable matter. *Natural Computing*, 17(4):723–741, 2018.
2. Rida A. Bazzi and Joseph L. Briones. Brief Announcement: Deterministic Leader Election in Self-organizing Particle Systems. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pages 381–386. Springer, 2018.
3. Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. A Markov chain algorithm for compression in self-organizing particle systems. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 279–288. ACM, 2016.
4. Joshua J. Daymude, Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. On the runtime of universal coating for programmable matter. *Natural Computing*, 17(1):81–96, 2018.
5. Joshua J. Daymude, Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Christian Scheideler, and Andréa W. Richa. Convex Hull Formation for Programmable Matter. *arXiv preprint*, 2018. `arXiv:1805.06149`.
6. Joshua J. Daymude, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Improved leader election for self-organizing programmable matter. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 127–140. Springer, 2017.
7. Joshua J. Daymude, Kristian Hinnenthal, Andréa W. Richa, and Christian Scheideler. Computing by Programmable Particles. In *Distributed Computing by Mobile Entities*, pages 615–681. Springer, 2019.
8. Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: amoebot–a new model for programmable matter. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pages 220–222. ACM, 2014.
9. Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication*, page 21. ACM, 2015.

10 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim 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.

11 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Universal coating for programmable matter. *Theoretical Computer Science*, 671:56–68, 2017.

12 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, Thim Strothmann, and Shimrit Tzur-David. Infinite object coating in the amoebot model. *arXiv preprint*, 2014. `arXiv:1411.2356`.

13 Zahra Derakhshandeh, Robert Gmyr, Thim Strothmann, Rida Bazzi, Andréa W. Richa, and Christian Scheideler. Leader election and shape formation with self-organizing programmable matter. In *International Workshop on DNA-Based Computers*, pages 117–132. Springer, 2015.

14 Giuseppe Antonio Di Luna, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Line recovery by programmable particles. In *19th International Conference on Distributed Computing and Networking, ICDCN 2018*, volume 133180, pages 1–10. Association for Computing Machinery, 2018.

15 Giuseppe Antonio Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. Shape Formation by Programmable Particles. In *21st International Conference on Principles of Distributed Systems*, 2017.

16 Reinhard Diestel. Graph theory. 2005. *Grad. Texts in Math*, 101, 2005.

17 Nicolas Gastineau, Wahabou Abdou, Nader Mbarek, and Olivier Togni. Distributed leader election and computation of local identifiers for programmable matter. *arXiv preprint*, 2018. `arXiv:1807.10461`.

18 Adrian Segall. Distributed network protocols. *IEEE transactions on Information Theory*, 29(1):23–35, 1983.

19 Tommaso Toffoli and Norman Margolus. Programmable matter: concepts and realization. *Physica D: Nonlinear Phenomena*, 47(1-2):263–272, 1991.