Brief Announcement: Deterministic Leader Election in Self-organizing Particle Systems

Rida A. Bazzi(✉) and Joseph L. Briones

Arizona State University, Tempe, AZ, USA
bazzi@asu.edu

Abstract. We consider the leader election problem in the geometric Amoebot model in which nodes have no unique identifiers and only share a common local sense of direction. Unlike other works, we consider the deterministic leader election problem for general connected systems. We propose a new deterministic leader election protocol that always succeeds in finding 1, 2, 3, or 6 leaders. We show that if the protocol does not elect a unique leader, deterministic leader election impossible for the system.

1 Introduction

Leader election is a fundamental problem that has been studied in both shared memory and message passing system models. It is a prototypical symmetry breaking problem [7]. The goal of leader election is to identify a unique member of the system as the leader. In anonymous systems, the requirement is for one unique member to self-identify as a leader and for other members to agree that a leader has been self-identified.

In this paper, we are interested in leader election in self-organizing particle systems, specifically the well studied Amoebot Model [3]. In this model, particles occupy cells in a hexagonal grid. They have finite memory, can communicate with adjacent particles, and can expand into unoccupied adjacent cells. The system of particles is assumed to be initially connected because there is no way to achieve coordination between different connected components without additional system assumptions [6]. Electing a leader can facilitate solving problems such as shape-formation [8], object coating [5] and system compression [1].

Leader election in the Amoebot model has been studied in the general case without restrictions on the connectedness of the system [2,4], but those solutions are probabilistic. The only deterministic solution for leader election is that of Di Luna et al. [8] who use deterministic leader election to solve the deterministic shape formation problem. While technically involved, their solution assumes that the particle system is simply connected which means that the unoccupied cells form a connected component. Their solution takes advantage of the fact that the shape has no holes. It starts with an initial *erosion* phase in which particles on the *corners* of the system eliminate themselves as candidate leaders. This

T. Izumi and P. Kuznetsov (Eds.): SSS 2018, LNCS 11201, pp. 381–386, 2018.
https://doi.org/10.1007/978-3-030-03232-6_25

phase ends with a unique leader or 2 or 3 candidate leaders. If there are 2 or 3 candidate leaders left, they form trees containing other particles in the system and compare these trees to each other to break the symmetry. If the trees are identical, it follows that the particle system has symmetry that makes deterministic leader election impossible. The approach of Di Luna et al. does not work in a system with holes because erosion does not work in the general case. The work of Di Luna et al. assumes no shared local sense of direction (chirality), but, once candidate leaders have been identified, achieves a common chirality using particle movement to beak symmetry.

The main contribution of this paper is a solution to the deterministic leader election problem in general connected systems. To overcome the limitation of the earlier work, we come up with a novel approach to determine a small number of leaders (1, 2, 3, or 6) on the unique outer boundary of the system. After the candidate leaders are determined, the solution proceeds as in [8]. Candidate leaders grow trees that are then compared to break symmetry. If breaking symmetry is not possible, then, like [8] we establish constructively that deterministic leader election is not possible. Unlike [8] in which candidate leaders are adjacent, in our setting, coordinating candidate leaders requires more care.

2 System Model

We consider the geometric Amoebot model [3] in which anonymous particles with finite memory occupy cells within a hexagonal lattice. Particles have the same chirality. Particles can occupy one cell (contracted) or two cells (expanded) and no cell can be occupied by more than one particle. Since our leader election algorithm does not involve any expansion, each particle has six ports ordered clockwise from port 0 through port 5, one port on each of the adjacent cells. The ports are used to communicate with other particles in adjoining cells and to sense if an adjacent cell is occupied by another particle or is empty. A communication edge between two particles consists of a pair of corresponding ports. For example, between cells A and B in Fig. 1, port 3 of A and port 4 of B form a communication edge between A and B. Two adjacent particles know the port numbers that form the edge between them. Particles communicate by writing to their local memory and reading the local memory of adjacent particles. This allows for a simple message passing between particles. We assume the particle system to be connected.

In the solution, we introduce six virtual nodes for each cell, one node per port. These nodes are represented in Fig. 1 by black dots at the vertices of the cells occupied by the particle. Port i is on the edge between node $i-1 \mod 6$ and node i. Nodes execute steps when they are activated by the scheduler which we assume to be completely asynchronous.

3 Leader Election

The algorithm has two main phases. In the first phase, a small number of candidate leaders are selected on the outer boundary of the system and in the second phase further reduction of this number is attempted. If the algorithm does not elect a unique leader in the second phase, the system must have symmetry that prevents deterministic leader election.

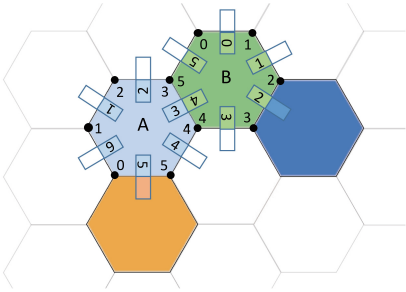

Fig. 1. Particle system surrounded by unoccupied cells

In the first phase, the algorithm starts by running separate instances of a *boundary leader election* algorithm on all the boundaries of the system (a node is on a boundary if it is adjacent to an empty cell). An instance of a boundary leader election algorithm is designed to work correctly if the participants in the election consist of all nodes of a boundary. On the inner boundaries, if any, it is guaranteed that no leader is elected. On the unique outer boundary, 1, 2, 3, or 6 candidate leaders are selected. Each leader also has what we call a *stretch*, a sequence of contiguous nodes, associated with it. If there is a unique leader, the algorithm terminates, but if there are multiple leaders, this means that the outer boundary has symmetry that prevents the deterministic election of a unique leader. This initial phase is the more involved phase and is done in a sequence of phases that are not strictly synchronized. Having a small number of leaders that can communicate around the outer boundary allows us to use the tree comparison approach of [8] in the second phase.

If at the end of the first phase there are multiple candidate leaders, each particle with a leader node tries to recruit as many particles as it can to form a tree with the particle itself as the root of the tree. This is the same as the approach of [8]. After all particles in the systems have joined a tree, each root compares its tree to the tree of the root to its *right* on the outer border according to an order relation. Every candidate leader then shares the results of these comparisons will all other candidate leaders on the boundary. If the results of all these comparison are equality, then there is symmetry in the system and deterministic leader election is not possible. If the result of one of the comparisons

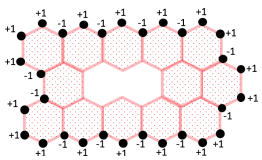

Fig. 2. Vertex labeling and initialization of stretches.

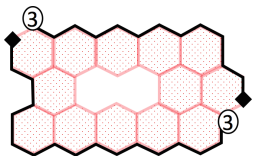

Fig. 3. Intermediate step with two stretches remaining.

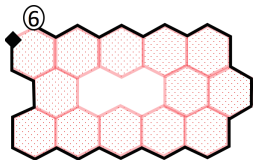

Fig. 4. Final configuration. Termination detected.

Algorithm 1. Stretch Expansion

```
1: function AttemptExpansion()
2:     ▷ s and s′ are two adjacent stretches. s′ is to the right of s.
3:     if s.count > s′.count ∧ (s.count + s′.count ≤ 6 ∧ s.count > 0) then
4:         Merge(s, s′)
5:     else if s.count = s′.count = 1, 2, 3, or 6 then
6:         if s ≡ s′ then                    ▷ if s and s′ are lexicographically equal
7:             DetectTermination()           ▷ initiate termination detection
8:         else if s > s′ then               ▷ if s is lexicographically greater than s′
9:             Merge(s, s′)
```

is inequality, then one or more candidate leaders are eliminated and the process is repeated with the remaining roots. This is repeated a constant number of times until either there is one unique remaining leader or there are multiple leaders who are all tied in the tree comparison. If there is a unique leader, we are done, otherwise, there is symmetry that prevents deterministic leader election. In what follows, we describe some of the details of the first phase.

The first phases starts by having each particle sense its surrounding to determine if one or more cells around it are unoccupied. A particle can be on more than one border, but each node can be on at most one border. When a particle has identified itself as part of the outer border and its successors and predecessors have been initialized, the particle labels its nodes with a unary label which is $+1$ for border nodes that belong to only one particle and -1 for border nodes that are shared between adjacent particles. This is illustrated in Fig. 2.

After labeling each node on the outer border, stretches attempt to expand to eliminate possible leaders. The leftmost node in a stretch is considered the leader (or head) of the stretch. The rightmost node in a stretch is called the tail of a stretch. Within a stretch, each node has a predecessor pointer and a successor pointer. The leader of the stretch maintains a counter which is equal to the sum of the unary labels of the nodes in the stretch. The counter value never exceeds the value 6. Initially, all nodes on the outer border are considered independent stretches, of size 1, with a respective counter equal to their unary label. All nodes are initialized to be both the head and tail of their stretch. Figures 3 and 4 illustrate stretches.

Algorithm 2. Termination Detection

```
1: function DetectTermination(s)
2:     terminate ← true
3:     for i ← 1, k/s.count do                                  ▷ k = 6/s.count
4:         s' ← s
5:         for j ← 1, i − 1 do
6:             s' ← s'.left                          ▷ Rotate to the stretch left of s'
7:             terminate ← (terminate ∧ (s'.count ≡ s.count))
8:         s' ← s'.left
9:         terminate ← (terminate ∧ (s'.count = s.count)) ∧ s' ≡ s
10:    return terminate
```

Stretches can expand by merging with adjacent stretches. When two stretches merge, the leader of the stretch on the left (s in Algorithm 1) becomes the leader of the resulting stretch and its new count is the sum of its old count and the count of the stretch being merged into. A merge is allowed only if the sum of the two counts is less than or equal to 6. We avoid deadlocks by placing an order relationship based on the count and lexicographic comparison between stretches. We require that the stretch s on the left has a positive count and either its count is larger than that of the stretch s' on the right (s') or the two counts are equal, but the sequence of unary labels of s is lexicographically larger than that of s'.

Finally, to detect termination, a stretch attempts to establish that the whole border on which it resides is covered with k identical stretches that have the same positive count ($k = 1, 2, 3$, or 6).

References

1. Cannon, S., Daymude, J.J., Randall, D., Richa, A.W.: A Markov chain algorithm for compression in self-organizing particle systems. In: Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, pp. 279–288. ACM (2016)
2. Daymude, J.J., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: Improved leader election for self-organizing programmable matter. In: Fernández Anta, A., Jurdzinski, T., Mosteiro, M.A., Zhang, Y. (eds.) ALGOSENSORS 2017. LNCS, vol. 10718, pp. 127–140. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72751-6_10
3. Derakhshandeh, Z., Dolev, S., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: Brief announcement: amoebot-a new model for programmable matter. In: Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures, pp. 220–222. ACM (2014)
4. Derakhshandeh, Z., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: 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, p. 21. ACM (2015)
5. Derakhshandeh, Z., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: Universal coating for programmable matter. Theor. Comput. Sci. **671**, 56–68 (2017)

6. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. Theor. Comput. Sci. **407**(1–3), 412–447 (2008)
7. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Inf. Comput. **88**(1), 60–87 (1990)
8. Di Luna, G.A., Flocchini, P., Santoro, N., Viglietta, G., Yamauchi, Y.: Shape formation by programmable particles. In: 21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, 18–20 December 2017, pp. 31:1–31:16 (2017)