Invited Paper: Asynchronous Deterministic Leader Election in Three-Dimensional Programmable Matter

Joseph L. Briones
Arizona State University
School of Computing and Augmented Intelligence
Tempe, AZ, USA
joseph.briones@asu.edu

Tishya Chhabra
Arizona State University
School of Computing and Augmented Intelligence
Tempe, AZ, USA
tishyac3.141@gmail.com

Joshua J. Daymude
Arizona State University
School of Computing and Augmented Intelligence
Biodesign Center for Biocomputing, Security and Society
Tempe, AZ, USA
jdaymude@asu.edu

Andréa W. Richa
Arizona State University
School of Computing and Augmented Intelligence
Tempe, AZ, USA
aricha@asu.edu

ABSTRACT

Over three decades of scientific endeavors to realize *programmable matter*, a substance that can change its physical properties based on user input or responses to its environment, there have been many advances in both the engineering of modular robotic systems and the corresponding algorithmic theory of collective behavior. However, while the design of modular robots routinely addresses the challenges of realistic three-dimensional (3D) space, algorithmic theory remains largely focused on 2D abstractions such as planes and planar graphs. In this work, we formalize the *3D geometric space variant* for the *canonical amoebot model* of programmable matter, using the *face-centered cubic (FCC) lattice* to represent space and define local spatial orientations. We then give a distributed algorithm for *leader election* in connected, contractible 2D or 3D geometric amoebot systems that deterministically elects exactly one leader in $O(n)$ rounds under an unfair *sequential* adversary, where n is the number of amoebots in the system. We then demonstrate how this algorithm can be transformed using the concurrency control framework for amoebot algorithms (DISC 2021) to obtain the first known amoebot algorithm, both in 2D and 3D space, to solve leader election under an unfair *asynchronous* adversary.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**; *Concurrent algorithms*; *Self-organization.*

KEYWORDS

programmable matter, leader election, three-dimensional

ACM Reference Format:
Joseph L. Briones, Tishya Chhabra, Joshua J. Daymude, and Andréa W. Richa. 2023. Invited Paper: Asynchronous Deterministic Leader Election in Three-Dimensional Programmable Matter. In *24th International Conference on Distributed Computing and Networking (ICDCN 2023), January 4–7, 2023, Kharagpur, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3571306.3571389

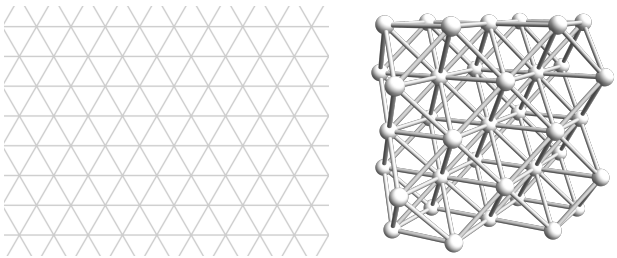
ICDCN 2023, January 4–7, 2023, Kharagpur, India

ACM ISBN 978-1-4503-9796-4/23/01.
https://doi.org/10.1145/3571306.3571389

1 INTRODUCTION

Since its inception [27], programmable matter has been envisioned as a material that can dynamically alter its physical properties based either on user input or autonomous sensing of the environment. Many strides have been made to realize this technology over the last several decades, both from the practical perspective of modular robotics and the algorithmic contributions of distributed computing theory. However, when it comes to realistic considerations of three-dimensional (3D), gravity-bound space, advances in robotics have outpaced their distributed computing counterparts. Modular, reconfigurable robotic systems such as Proteo [31], SlidingCube [14], 3D M-Blocks [24], RollingSphere [20], FireAnt3D [25], FreeBOT [18, 19], and 3D Catoms [23, 26] routinely address engineering challenges both in individual module design (such as binding and locomotion) and in collective reconfiguration (such as gravity stability) that are inherent to 3D environments. Besides a few notable exceptions [29, 30], the vast majority of abstract models of mobile robots and programmable matter treat space as two-dimensional (2D) planes or planar graph structures [2, 4, 9, 15, 21, 22, 28], simplifying their assumptions but limiting their application to practical domains.

Our goal is to move theoretical programmable matter research towards the 3D reality by extending the established *amoebot model* of programmable matter [8, 9]. Research using the amoebot model has historically assumed 2D discretizations of space, most commonly the "geometric" triangular lattice (Figure 1a). Under this treatment of space, amoebot algorithms have been developed for a myriad of problems including leader election, shape formation, object coating and enclosure, bridging, and more (see [5, 7, 8] for an overview of results). The recent *canonical amoebot model* [8] systematized the many disparate assumptions appearing in these works into categories, each with a set of "assumption variants" of varying strengths. In this paper, we formalize the *3D geometric space*



(a) (b)

Figure 1: *The Geometric Space Variants.* (a) The 2D triangular lattice G_Δ. (b) The 3D face-centered cubic (FCC) lattice G_{FCC}.

variant for the canonical amoebot model that discretizes space as the *face-centered cubic (FCC) lattice* (Figure 1b).

Lattice representations of modular robots and programmable matter typically assume either a cubic lattice corresponding to cubic or spherical modules [19, 20, 24, 29] or an FCC lattice corresponding to (quasi-)spherical or rhombic-dodecahedral modules [17, 23, 26]. The FCC lattice is a natural 3D generalization of the 2D triangular lattice already used for many amoebot algorithms; in fact, it can be decomposed into layers of triangular lattices and thus remains compatible with existing results for 2D amoebot systems. Also, maintaining module connectivity during movements is easier in an FCC lattice than in a cubic one (see, e.g., [23]).

Using this new 3D geometric space variant, we revisit the classical problem of *leader election*, defined formally in Section 4. We demonstrate that among the many existing algorithms for leader election in 2D amoebot systems [1, 6, 10–13, 16], the *erosion-based algorithm* of Di Luna et al. [11] extends naturally to 3D—surprisingly, without any cost to runtime. Although erosion-based election in 3D may seem simple at first glance, its analysis requires new, nontrivial topological arguments specific to the 3D setting. Our algorithm elects exactly one leader in any connected, contractible amoebot system (defined formally in Section 3) within $O(n)$ rounds under an unfair sequential adversary, where n is the number of amoebots in the system. We thus achieve similar guarantees as the state-of-the-art algorithm for 3D leader election by Gastineau et al. [17], with two important differences: (1) we consider all 24 possible amoebot orientations in 3D achievable by rotation or reflection while Gastineau et al. only consider eight, and (2) we break symmetry using local comparisons between neighbors' orientations while Gastineau et al. assume 2-neighborhood vision. We further show that our algorithm is compatible with the *concurrency control framework* for amoebot algorithms [8], implying that it can be transformed into an algorithm with equivalent behavior that remains correct even under an unfair asynchronous adversary.

Our Contributions. Our main contributions are summarized as:

- A formalization of the *3D geometric space variant* for the canonical amoebot model using the FCC lattice to discretize space and define amoebots' spatial orientations (Section 3).
- A deterministic amoebot algorithm that solves leader election in both 2D and 3D geometric space for connected, contractible systems under an unfair sequential adversary within $O(n)$ rounds, where n is the number of amoebots in the system (Sections 5–6).
- An application of the concurrency control framework for amoebot algorithms [8] that yields the first known amoebot algorithm, in both 2D and 3D space, to solve leader election under an unfair asynchronous adversary (Section 7).

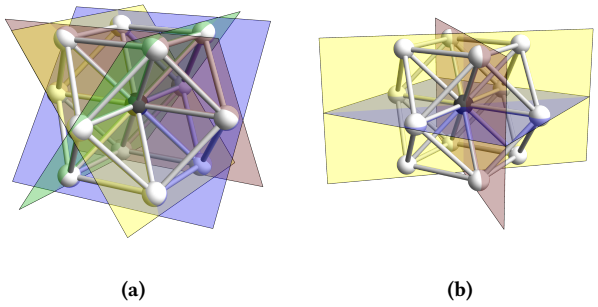
2 THE AMOEBOT MODEL

We begin by describing the features of the canonical amoebot model that will be used in this work; a deeper description of the model and its rationale can be found in [8]. In the canonical amoebot model, programmable matter consists of individual, homogeneous computational elements called *amoebots*. The structure of an amoebot system is represented as a subgraph of an infinite, undirected graph $G = (V, E)$ where V represents all relative positions an amoebot can occupy and E represents all atomic movements an amoebot can make. Each node in V can be occupied by at most one amoebot at a time. There are many possible assumption variants one could make about space; here, we consider the *2D geometric* variant which assumes $G = G_\Delta$, the triangular lattice (Figure 1a), and the presently introduced *3D geometric* variant which assumes $G = G_{\text{FCC}}$, the face-centered cubic lattice (Figure 1b).

In this work, all amoebots remain CONTRACTED, each occupying a single node in V; other works also consider EXPANDED amoebots that occupy a pair of adjacent nodes in V. Each amoebot keeps a collection of ports—one for each edge incident to the node it occupies—that are labeled consecutively according to its own local, persistent *orientation*. An amoebot's orientation is defined according to space variant-specific information; we define orientation for our lattices of interest in Section 3. Two amoebots occupying adjacent nodes are said to be *neighbors*. Although each amoebot is *anonymous*, lacking a unique identifier, an amoebot can locally identify its neighbors using their port labels. In particular, amoebots A and B connected via ports p_A and p_B are each assumed to know one another's orientations and labels for p_A and p_B.

Each amoebot has memory whose size is a model variant; here we assume *constant-size* memories. An amoebot's memory consists of two parts: a persistent *public memory* that is only accessible to an amoebot algorithm via communication operations (defined next), and a volatile *private memory* that is directly accessible by amoebot algorithms for temporary variables, private computation, etc. *Operations* define the programming interface for amoebot algorithms to communicate, move, and control concurrency that are, in reality, implemented via message passing (see [8] for details). Our algorithm for leader election only makes use of the communication operations CONNECTED, READ, and WRITE.

- The CONNECTED operation tests the presence of neighbors. CONNECTED(p) returns TRUE if and only if there is a neighbor connected via port p.
- The READ and WRITE operations exchange information in public memory. READ(p, x) issues a request to read the value of a variable x in the public memory of the neighbor connected via port p while WRITE(p, x, x_{val}) issues a request to update its value to x_{val}. If $p = \perp$, an amoebot's own public memory is accessed instead of a neighbor's.



(a) (b)

Figure 2: *Lattice Decompositions of* G_{FCC}. A node of G_{FCC} (black) viewed as the intersection of (a) four non-parallel triangular lattices or (b) three orthogonal square grids.

Amoebot algorithms are defined as sets of *actions*, each of the form $\langle label \rangle : \langle guard \rangle \rightarrow \langle operations \rangle$. An action's *label* specifies its name. Its *guard* is a Boolean predicate determining whether an amoebot A can execute it based on the ports A has connections on—i.e., which nodes adjacent to A are (un)occupied—and information from the public memories of A and its neighbors. An action is *enabled* for an amoebot A if its guard is true for A, and an amoebot is *enabled* if it has at least one enabled action. An action's *operations* specify the finite sequence of operations and computation in private memory to perform if this action is executed.

An amoebot is *active* if it is currently executing an action and is *inactive* otherwise. The model assumes an *adversary* controls the timing of amoebot activations and the resulting action executions, whose *concurrency* and *fairness* are assumption variants. In this work, we consider two concurrency variants: *sequential*, in which at most one amoebot can be active at a time; and *asynchronous*, in which any set of amoebots can be simultaneously active. We consider the most general fairness variant: *unfair*, in which the adversary may activate any enabled amoebot.

An amoebot algorithm's time complexity is evaluated in terms of *rounds* representing the time for the slowest continuously enabled amoebot to execute a single action. Let t_i denote the time at which round $i \in \{0, 1, 2, \ldots\}$ starts, where $t_0 = 0$, and let $\mathcal{E}_i$ denote the set of amoebots that are enabled or already executing an action at time t_i. Round i completes at the earliest time $t_{i+1} > t_i$ by which every amoebot in $\mathcal{E}_i$ either completed an action execution or became disabled at some time in $(t_i, t_{i+1}]$. Depending on the adversary's concurrency, action executions may span more than one round.

3 THE THREE-DIMENSIONAL (3D) GEOMETRIC SPACE VARIANT

We now formally define the *3D geometric space variant* for the canonical amoebot model, one of the main contributions of this work. This variant assumes space is represented by the *face-centered cubic (FCC) lattice*, G_{FCC} (Figure 1b). Just as the triangular lattice G_Δ used by the 2D geometric space variant can be viewed as the adjacency graph of the closest circle packing or as the dual of the hexagonal tiling, G_{FCC} can be viewed as the adjacency graph of the sphere packing that minimizes unoccupied volume or as the

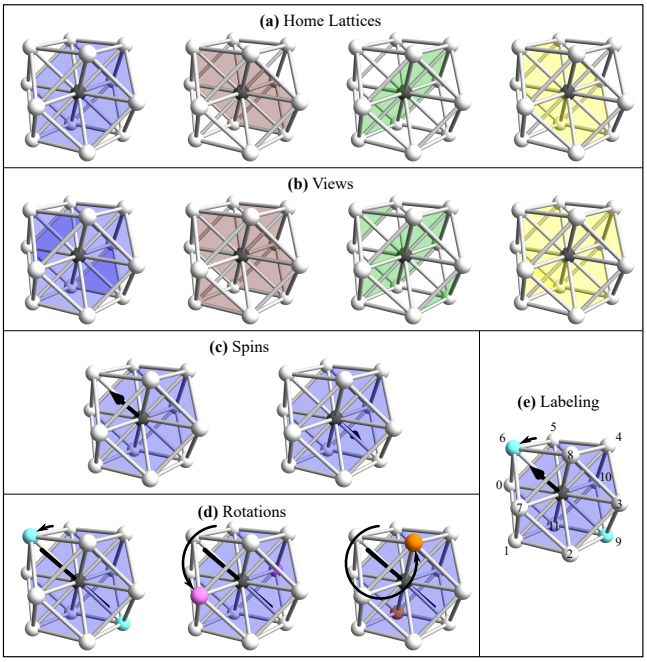


Figure 3: *Amoebot Orientations*. (a)–(d) The components of an amoebot's orientation. For brevity, only the spins and rotations of the blue home lattice are shown in (c) and (d). (e) An example port labeling.

dual of the rhombic-dodecahedral tessellation. Each node in G_{FCC} has degree 12 and can be viewed as the intersection of four infinite, non-parallel triangular lattices (Figure 2a) or as the intersection of three orthogonal square grids (Figure 2b). Thus, in 3D geometric space, a contracted amoebot has 12 neighbors and an expanded amoebot has at most 18.

An amoebot's *orientation* represents all the ways its local sense of space can be rotated or reflected while respecting the underlying spatial structure. In G_Δ, an amoebot's orientation is represented as a *direction* indicating which incident lattice edge it thinks of as "north" and a *chirality* establishing the clockwise vs. counterclockwise ordering of its incident edges. We generalize orientation in G_{FCC} using *view*, *spin*, and *rotation* as defined below. Different model variants may assume that amoebots share all, some, or none of their views, spins, and rotations in common. This work assumes *assorted* orientations, meaning the amoebots are not guaranteed to share any aspect of their orientations.

The *home lattice* of an amoebot is one of the four infinite triangular lattices that contain the node the amoebot occupies (Figure 3a). An amoebot's *view* is the triangular lattice decomposition of G_{FCC} containing the amoebot's home lattice (Figure 3b). A view can alternatively be defined as the set of triangular lattices whose planar embeddings in 3D space are non-intersecting, each orthogonal to the same two anti-parallel vectors. An amoebot's *spin* defines the "top" and "bottom" sides of the amoebot's home lattice; formally, it is the differentiation of the home lattice's two orthogonal vectors as positive and negative (Figure 3c). Having fixed this spin vector, any node's incident edges contained in the amoebot's view can be

Table 1: *Comparison of Amoebot Algorithms for Leader Election.* Algorithms are organized chronologically by first publication. Gastineau et al. [17] details two separate leader election algorithms; the row with "#2" is specific to its "Algorithm 2" which is marked with a * to denote its non-standard assumption of 2-neighborhood vision. When $k \in \mathbb{Z}_+$ appears in the number of leaders elected, it refers to the amoebot system being k-symmetric. For the runtime bounds, L^* denotes the length of the longest system boundary, L denotes the length of the system's outer boundary, D is the diameter of the system's initial configuration, and n denotes the number of amoebots in the system. The runtime terms r and $mtree$ are specific to [16], and it can be shown that $r + mtree$ is both $\Omega(D)$ and $O(n)$.

Algorithm	Space	Assorted Orientation	Adversary	Deterministic	Allows "Holes"	Stationary	Leaders Elected	Runtime
Derakhshandeh et al. [10]	2D	Direction	Strong Seq.	✗	✓	✓	1	$O(L^*)$ exp.
Daymude et al. [6]	2D	Direction	Strong Seq.	✗	✓	✓	1, w.h.p.	$O(L)$ w.h.p.
Di Luna et al. [11]	2D	Both	Sync.	✓	✗	✓	$k \leq 3$	$O(n^2)$
Gastineau et al. [16]	2D	Direction	Strong Seq.	✓	✗	✓	1	$O(r + mtree)$
Emek et al. [13]	2D	Both	Strong Seq.	✓	✓	✗	1	$O(Ln^2)$
Bazzi and Briones [1]	2D	Direction	Weak Seq.	✓	✓	✓	$k \leq 6$	$O(n^2)$
Dufoulon et al. [12]	2D	Direction	Strong Seq.	✓	✓	✗	1	$O(L + D)$
Gastineau et al. #2* [17]	3D	Spin/Rotation	Strong Seq.	✓	✗	✓	1	$O(n)$
This Paper	2&3D	All	Strong Seq.	✓	✗	✓	1	$O(n)$
This Paper + [8]	2&3D	All	Async.	✓	✗	✓	1	**?**

listed in clockwise order according to the right-hand rule. The final component of an amoebot's orientation is its *rotation* about its spin vector, of which there are three that agree with G_{FCC} (Figure 3d). A rotation can alternatively be defined as one of the three axes in G_{FCC} that contain the amoebot's node but are not contained in the amoebot's home lattice.[1]

As in the 2D geometric space variant, an amoebot uses its orientation in 3D geometric space to define a consistent labeling of its ports (i.e., incident edges). In Figure 3e, we illustrate one possible labeling for a contracted amoebot. The precise labeling scheme is immaterial so long as all amoebots label their ports consistently as a function of their (potentially differing) orientations.[2]

We can connect the 2D and 3D geometric space variants as follows. The single triangular lattice G_Δ in 2D geometric space can be thought of as any of the triangular lattices contained in G_{FCC}; i.e., all amoebots in a 2D system have the same home lattice (and thus the same view) in a 3D system. An amoebot's chirality in a 2D system plays the same role as its spin in a 3D system, defining "up" vs. "down" and clockwise vs. counterclockwise relative to the home lattice. Finally, an amoebot's direction in a 2D system functions analogously to rotation about its spin vector in a 3D system, with one discrepancy: due to the constraints of the underlying lattices, a 2D system allows six possible 60° rotations while a 3D system allows three possible 120° rotations.

Finally, we define two spatial properties of amoebot systems that will be used throughout the remaining sections. An amoebot system is *connected* if the lattice nodes occupied by amoebots induce a single connected component. We are also concerned with the notion of "holes" in amoebot systems. In general topology, a *hole* is any topological structure that prevents an object (or space) from being continuously shrunk to a single point; an object is *contractible* if it does not contain holes. To apply this definition to amoebot systems, recall that the dual of G_{FCC} (resp., G_Δ) is the rhombic dodecahedral (resp., hexagonal) tessellation of space. Given a 3D (resp., 2D) amoebot system, its *lattice dual representation* is the closed union of all solid rhombic dodecahedrons (resp., hexagons) in the lattice's dual corresponding to lattice nodes occupied by amoebots. An amoebot system contains a hole if and only if its lattice dual representation contains a (topological) hole and is contractible otherwise.

4 AMOEBOT LEADER ELECTION

An algorithm *solves the leader election problem* if for any connected system of initially contracted amoebots with well-initialized memories, eventually a single amoebot irreversibly declares itself the system's leader and no other amoebot ever does so. A leader's ability to break symmetry and coordinate the system via broadcasts makes it a powerful primitive for other amoebot algorithms, spurring a flurry of recent research on amoebot algorithms for leader election [1, 4, 6, 10–13, 16, 17]. Table 1 compares these existing algorithms, their assumptions, and their outcomes to our own, though we treat the D'Angelo et al. algorithm [4] and "Algorithm 1" of Gastineau et al. [17] separately due to their strong, one-off assumptions. The common assumptions and outcomes are:

- *Space.* Nearly all amoebot algorithms for leader election assume the 2D geometric space variant. Recently, Gastineau et al. [17] introduced a pair of algorithms for amoebot leader election on G_{FCC}, much like our own 3D geometric space variant, but with stronger assumptions (see below).
- *Orientation.* Recall that in 2D, amoebot orientation is defined as a direction and chirality; analogously, 3D orientations are

[1] An amoebot's orientation could be defined more succinctly as a pair of vectors originating at the amoebot's node: one that is orthogonal to the amoebot's home lattice and pointing to its "top" side (defining view and spin), and a second that points to a neighboring node outside the amoebot's home lattice (defining rotation).

[2] This is strictly more general than the port labeling schemes allowed in the formulation of 3D space by Gastineau et al. (see Sections 2.1 and 2.2.1 of [17]). Their formulation assumes all amoebots share a common sense of "layers" analogous to our views, but for a square lattice decomposition of G_{FCC} instead of a triangular one. With square "home layers", there are two possible spins ("up" vs. "down") and four possible rotations, yielding eight total orientations. Our formulation considers all 24.

a view, spin, and rotation. In 2D, the algorithms of Di Luna et al. [11] and Emek et al. [13] allow fully assorted orientations while the rest assumed common chirality. In 3D, "Algorithm 2" of Gastineau et al. [17] assumes common (square) views while our algorithm allows fully assorted orientations.

- *Adversary/Scheduler.* Until the canonical amoebot model was introduced, most algorithms assumed a sequential scheduler under which at most one amoebot could be active at a time. Strong sequential schedulers allow each amoebot to read, write, and move in one atomic action. Weak sequential schedulers used by Bazzi and Briones [1] are more general, ensuring only that reading, writing, and moving are each individually atomic but may not necessarily be combined into a single atomic action. Di Luna et al. [11] assume a synchronous scheduler that may activate arbitrary subsets of amoebots concurrently, but only in discrete stages. This paper starts with a strong sequential scheduler but ultimately considers the most general asynchronous adversary that allows for arbitrary concurrency among amoebot actions.
- *Deterministic vs. Randomized.* Randomization is a classical technique for symmetry breaking, but incurs a failure probability (with respect to correctness, runtime, or both) that is not present in deterministic algorithms. The original Derakhshandeh et al. algorithm [10] and its improvement by Daymude et al. [6] are both randomized while the rest, including our present algorithm, are deterministic.
- *Connectivity and Holes.* All existing algorithms assume connected initial system configurations. The algorithms of Di Luna et al. [11] and Gastineau et al. [16] further assume their 2D initial configurations are hole-free. Our algorithm analogously assumes its 3D initial configurations are contractible (as defined in Section 3), which is a necessary but likely insufficient condition for the "electable" configurations assumed by "Algorithm 2" of Gastineau et al. [17].
- *Movement.* Emek et al. [13] and Dufoulon et al. [12] utilize amoebots' movement capabilities as a mechanism for symmetry breaking. All other algorithms, including ours, are *stationary*, relying only on communication to elect a leader.
- *Number of Leaders Elected.* Due to symmetry in the initial system configuration, some of the stationary deterministic algorithms elect a constant number of leaders instead of a unique one. In particular, for k-symmetric system configurations, the Di Luna et al. algorithm [11] elects $k \in \{1, 2, 3\}$ leaders and the Bazzi and Briones algorithm [1] elects $k \in \{1, 2, 3, 6\}$. All other algorithms, including ours, elect a unique leader.
- *Runtime.* All existing algorithms' runtime bounds are given in sequential rounds, where a round ends when each amoebot has been activated at least once. Our analysis uses a comparable but more technical definition of a round that extends to any concurrency assumption and focuses on the behavior of enabled amoebots (see Section 2). Among the deterministic algorithms, only the Dufoulon et al. algorithm [12] achieves a faster runtime than ours.

D'Angelo et al. [4] introduced the SILBOT model which is inspired by the amoebot model and aims to study what collective behaviors are possible when robots cannot communicate via messages or memory accesses. Under this seemingly challenging model, they show how to deterministically elect up to three leaders (due to symmetry) if the 2D system configuration is connected and hole-free—and if the robots can sense the presence and shape of robots in their 2-neighborhood on G_Δ, a strong assumption that no other amoebot algorithm makes. They then additionally show how the algorithm can be generalized to connected configurations with holes if each robot can sense which unoccupied nodes are holes and which belong to the outside of the system. This very strong assumption is unique to SILBOT, resolving the challenging "inner-outer boundary problem" [6, 10] by simply granting the requisite knowledge to robots a priori. Our algorithm makes no such assumptions.

The pair of Gastineau et al. algorithms for leader election in 3D amoebot systems are the most relevant to ours since they also use G_{FCC} to discretize space [17]. However, their first algorithm assumes all amoebots have the same 3D orientation and $O(n \log n)$ memory, where n is the number of amoebots in the system. Their second algorithm addresses some of these concerns by assuming orientations with assorted spins and rotations but common views and constant-size memory, but allows amoebots to view "extended neighborhoods" that include nodes at distance 2. Our algorithm achieves the same leader election guarantees without these assumptions, using only constant-size memory and local comparisons between 1-neighbors' orientations for symmetry breaking.

5 3D LEADER ELECTION BY EROSION

Our algorithm for leader election in 3D amoebot systems is an extension of the "lattice consumption" algorithm for 2D systems by Di Luna et al. [11]. In the lattice consumption algorithm, all amoebots are initially eligible for leader candidacy. When activated, an eligible amoebot uses certain rules regarding the number and relative positions of its eligible neighbors to decide whether to *erode*, revoking its candidacy without disconnecting the set of eligible amoebots or introducing a hole. Assuming the initial configuration was connected and hole-free, Di Luna et al. [11] proved that under a synchronous scheduler, erosion would eventually reduce the system to 1, 2, or 3 candidate leaders depending on the system's symmetry.

Our algorithm generalizes this approach to the 3D geometric space variant by defining rules for erosion based on 3D neighborhoods. It deterministically elects a unique leader for connected, contractible systems by leveraging the sequential adversary to break symmetry (Section 6). We then lift this strong timing assumption to the asynchronous setting—the most general of all concurrency assumptions—using the lock-based concurrency control framework for amoebot algorithms [8] (Section 7).

Algorithm 1 details our algorithm's pseudocode and Table 2 summarizes each amoebot's local variables. Initially, the system has no leader and all amoebots exist in a special null candidacy state. Every amoebot's first activation executes the Setup action which sets the amoebot as a candidate and informs its neighbors of its candidacy. Once all neighbors of a given candidate amoebot A have also done their setup actions, the Erode action becomes enabled for A whenever A satisfies an erosion rule. When executing Erode, A revokes its candidacy and informs its neighbors that it did so. As we will prove in the next section, repeated executions of the

Algorithm 1 Leader Election by Erosion for Amoebot A

```
1: Setup : (A.cand = null) →
2:     Write(⊥, cand, true).                    ▷ Become a candidate.
3:     for each port p of A do                  ▷ Inform neighbors of candidacy.
4:         if Connected(p) then
5:             Let p′ be the neighbor's port connected to port p.
6:             Write(p, nbrcand(p′), true).
7: Erode : (A.cand = true) ∧ (∀B ∈ N(A), B.cand ≠ null) ∧ CanErode() →
8:     Write(⊥, cand, false).                   ▷ Revoke candidancy.
9:     for each port p of A do                  ▷ Inform neighbors of erosion.
10:        if Connected(p) then
11:            Let p′ be the neighbor's port connected to port p.
12:            Write(p, nbrcand(p′), false).
13: DeclareLeader : (A.cand = true) ∧ (∀B ∈ N(A), B.cand = false) →
14:    Write(⊥, leader, true).
15: function CanErode( )
16:    return true if and only if:
           • Rule 1: A has exactly one candidate neighbor; or
           • Rule 2: A has two to five candidate neighbors and these
             neighbors' positions induce a connected subgraph; or
           • Rule 3: A has exactly two candidate neighbors that have a
             common candidate neighbor such that these four candidates
             induce a square in G_FCC.
```

Table 2: *Algorithm Notation.* The domain, initialization, and description of the local variables used in the leader election algorithm by an amoebot A.

Variable	Domain	Init.	Description
leader	{true, false}	false	true iff A is the unique leader
cand	{null, true, false}	null	After first activation, true iff A is a candidate
nbrcand(p)	{true, false}	false	true iff A has a candidate neighbor on port p

Erode action eventually reduce the system to a single candidate that, upon finding no candidate neighbors, elects itself as leader in the DeclareLeader action.

It remains to specify the erosion rules for 3D geometric systems. We formally specify these rules below and visualize them in Figure 4. For the sake of clarity, we represent the collection of these rules in Algorithm 1 as a function CanErode that returns true if and only if the calling amoebot A satisfies one of the following erosion rules:

Rule 1. A has exactly one candidate neighbor (Figure 4a).

Rule 2. A has two to five candidate neighbors, and these neighbors' positions induce a connected subgraph (Figure 4b).

Rule 3. A has exactly two candidate neighbors that have a common candidate neighbor such that these four candidates induce a square in G_{FCC} (Figure 4c).

As we will show in Section 6, any connected and contractible system of at least two candidate amoebots contains at least one candidate A satisfying one of these three rules. We can further show that the "erosion" of A does not violate the connectivity or contractibility of the remaining candidate structure, ensuring that the system eventually converges to exactly one leader amoebot.

Rules 1 and 2 can be evaluated using only the local port labels of A that are connected to candidate neighbors—which can be obtained using Read operations on neighbors' cand variables—and some basic information about the structure of G_{FCC}. However, Rule 3 covers a special case that is specific to G_{FCC}. If four candidates A, B, C, and D form a square in which each candidate is adjacent to exactly two others[3] but not to the third that is "catty-corner" to it, then none of these four candidates can determine whether erosion will disconnect the candidate structure when using only the positions of their candidate neighbors. For A, safe erosion hinges on the existence of the catty-corner candidate C: if A erodes, then B and D remain connected if and only if C exists.

Instead of assuming this problem away by giving the amoebots 2-neighborhood vision, as "Algorithm 2" of Gastineau et al. [17] does, we address this problem locally using the nbrcand variables. An amoebot A has A.nbrcand(p) = true if and only if A has a neighbor candidate connected via port p; these are set to true by neighbors becoming candidates during their Setup actions and then are reset to false by eroding candidate neighbors during their Erode actions. Maintaining these variables allows an amoebot A that might be in a square to Read the corresponding nbrcand variable of its candidate neighbors B or D to check for the existence of the catty-corner candidate C. Importantly, this uses the amoebot model's assumption that neighboring amoebots know one another's orientations, and thus A can translate the adjacency it wants to check into the correct port label from the perspective of B or D.

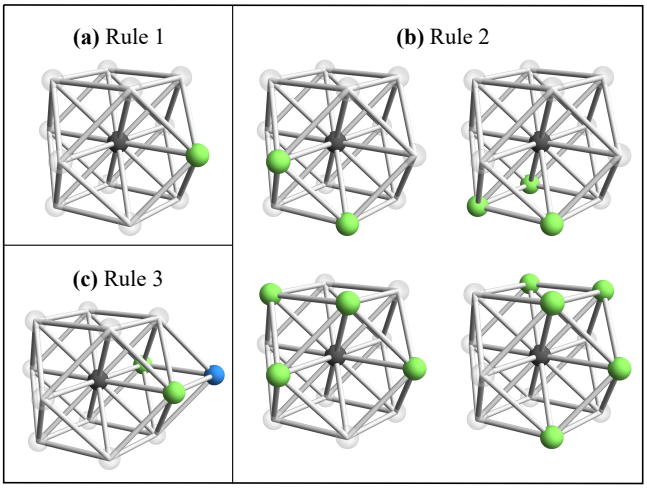


Figure 4: *Erosion Rules.* Example configurations of neighboring candidates (green) for which an amoebot A (black) can safely erode. (a)–(b) A can erode if it has one to five neighbors that induce a connected subgraph. (c) A can only erode if the catty-corner candidate neighbor (blue) exists.

6 SEQUENTIAL ANALYSIS

We first analyze our algorithm under an unfair sequential adversary; i.e., when at most one amoebot can be active per time and the adversary may activate any enabled amoebot. We will extend these results to an unfair asynchronous adversary allowing for arbitrary

[3] Adjacency in G_{FCC} is taken by (2D) face adjacency; note that amoebots A, B, C and D actually do intersect in the rhombic dodecahedral tessellation at exactly one vertex point.

concurrency in the next section. We first prove our algorithm never violates a safety condition.

Lemma 6.1. *Let S be a connected and contractible amoebot system on G_{FCC}. If amoebot A satisfies an erosion rule with respect to S, then $S - A$ is also connected and contractible.*

Proof. Suppose to the contrary that there exists an amoebot A in S that satisfies an erosion rule but $S - A$ is either disconnected or not contractible. Suppose first that $S - A$ is disconnected, i.e., there exist amoebots $\{B, C\} \not\ni A$ such that a simple path from B to C exists in the subgraph induced by S but not in the subgraph induced by $S - A$. Any (B, C)-path $\mathcal{P}$ in S must necessarily must have contained A and, by extension, two distinct neighbors N_1 and N_2 of A. Thus, A could not have eroded because it had exactly one neighbor in S (Rule 1). Moreover, A could not have eroded because it had between two and five neighbors in S that themselves induced a connected subgraph (Rule 2) since the path $\mathcal{P} = (B, \ldots, N_1, A, N_2, \ldots, C)$ could be augmented using the connectivity of N_1 and N_2 as $(B, \ldots, N_1, \ldots, N_2, \ldots, C)$, a (B, C)-path in $S - A$. So A must have eroded as a result of Rule 3; w.l.o.g., suppose the clockwise order of amoebots in the square is (A, N_1, X, N_2). But A could only have satisfied Rule 3 if its catty-corner amoebot X existed in S, implying the path $(B, \ldots, N_1, X, N_2, \ldots, C)$ exists in $S - A$. Therefore, in all cases, $S - A$ must be connected.

So suppose instead that $S - A$ is connected but is not contractible; i.e., $S - A$ contains a hole. Since S is contractible by supposition, the hole in $S - A$ must correspond to exactly the position of A; i.e., the lattice dual representation replacing the neighbors of A with rhombic dodecahedrons must contain a topological hole. One can prove using exhaustive search that no configuration of at most five neighbors has a corresponding closed union of rhombic dodecahedrons containing a topological hole.[4] Since all erosion rules involve A having between one and five neighbors in S, it is impossible for A to both satisfy an erosion rule and create a hole by eroding, a contradiction. □

We next leverage this safety condition to prove progress.

Lemma 6.2. *Any connected and contractible amoebot system S on G_{FCC} comprising at least two amoebots contains an amoebot that satisfies an erosion rule with respect to S.*

Proof. Argue by induction on $|S|$, the number of amoebots in S. When $|S| = 2$, the fact that S is connected implies that the two amoebots in S each have each other as their only neighbor in S. Thus, both of these amoebots satisfy Rule 1.

So suppose that $|S| > 2$ and that, for all $2 \leq k < |S|$, any connected and contractible amoebot system on G_{FCC} comprising k amoebots contains an amoebot satisfying an erosion rule. We decompose the amoebots of S into two types: "chain" amoebots whose removal would disconnect S, and those belonging to the remaining 2-connected components. A *chain* is a maximal set of amoebots $A_1, \ldots, A_\ell$ in S satisfying one of the following cases: (i) if $\ell > 2$, then for all $1 < i < \ell$, (A_{i-1}, A_i) and (A_i, A_{i+1}) are the only edges in S incident to A_i, $(A_1, A_\ell) \notin S$, and no other chain can exist with endpoints A_1 and A_ℓ, (ii) if $\ell = 2$, then $(A_1, A_2) \in S$ and no edge of S exists between $\{A_1\} \cup N_S(A_1)$ and $\{A_2\} \cup N_S(A_2)$, or (iii) if $\ell = 1$, then $N_S(A_1)$ is disconnected. The final condition in (i) regarding the uniqueness of a chain with given endpoints follows from the squares that exist in G_{FCC}. The lattice dual representation of a square is four rhombic dodecahedrons that intersect at a single point; thus, no one chain between catty-corner endpoints can individually disconnect S and thus are omitted. No other such "parallel chains" can exist in S because it is contractible.

Consider the graph composed of (i) all chain amoebots $\mathcal{C}$ in S and (ii) each connected component of $S - \mathcal{C}$ contracted to a single node. Since S is contractible, this graph must be a tree. Consider any leaf of this tree. If this leaf corresponds to an amoebot A^* with exactly one neighbor in S, then A^* satisfies Rule 1. Otherwise, the leaf corresponds to a connected component S' of $S - \mathcal{C}$ with at least two amoebots. Since S' is a leaf in the tree, there exists a unique chain amoebot $C \in \mathcal{C}$ connecting S' to the rest of S.

The remainder of this proof identifies an amoebot $A^* \neq C$ in S' that satisfies erosion Rule 2 or 3. If there exists an amoebot A^* in S' such that $N_{S' \cup \{C\}}(A^*) = \{X, Y\}$ and there exists an amoebot Z such that A^*, X, Y, and Z form an induced (chordless) square in G_{FCC}, then A^* satisfies erosion Rule 3.

Otherwise, if an amoebot satisfying Rule 3 cannot be found in S', we show there must exist an amoebot A^* in S' satisfying Rule 2. We do so by constructing an auxiliary (not necessarily convex) polyhedron P from the nodes of G_{FCC} occupied by amoebots in $S' \cup \{C\}$ that is contractible. Let P be the polyhedron in the 3D embedding of G_{FCC} that contains all the nodes occupied by $S' \cup \{C\}$ and whose faces are composed of (unions of) external triangles induced by $S' \cup \{C\}$, where an external triangle induced by $S' \cup \{C\}$ is a triangle whose three vertices are mutually adjacent nodes occupied by amoebots in $S' \cup \{C\}$ and where at least one of its vertices is adjacent to a node not occupied by $S' \cup \{C\}$. Note that the fact that the union of rhombic dodecahedrons corresponding to $S' \cup \{C\}$ is contractible, which is true by assumption, does not immediately imply that P will also be contractible, since by construction of P, P might possibly contain an "exposed" chordless square cycle of G_{FCC} on its surface whose nodes do not share a common neighbor: This is not possible, however, since if such a cycle existed in P, there would exist an amoebot A^* satisfying Rule 3 – i.e., an amoebot A in S' such that $N_{S' \cup \{C\}}(A) = \{X, Y\}$ and there exists an amoebot Z such that A, X, Y, Z form an induced (chordless) square cycle in G_{FCC}.

The *internal angle* of a face F at its vertex v is given by the angle internal to F defined by the two edges of F that contain vertex v. The *total internal angle* of a vertex v is the sum of internal angles of all faces incident to v, and its *external angle* is given by 2π minus its total internal angle:

$$\text{external-angle}(v) = 2\pi - \sum_{i \in k} \alpha_i,$$

where $\alpha_1, \ldots, \alpha_k$ are the interior angles of faces $F_1, \ldots, F_k$ incident to v. The *Euler characteristic* on closed surfaces is given by

$$\chi = 2 - 2g,$$

where g is the genus of the surface. Since P was shown to be contractible, its genus is $g = 0$ and its Euler characteristic is $\chi = 2$. From

[4] We verified this statement using a simple Mathematica program that enumerates each neighborhood polyhedron of at most five neighboring rhombic dodecahedrons and verifies that its genus is zero; see the link at the end of this paper.

the proof of the discrete version of the Gauss–Bonnet theorem [3],

$$\sum_{v \in P} \text{external-angle}(v) = 2\pi\chi.$$

Thus, the sum of external angles of all vertices of P is equal to 4π.

The external angle of any vertex can be at most 2π, so there must exist at least two vertices of P whose external angles are strictly positive. Since C is the unique chain amoebot in $S' \cup \{C\}$, there must exist at least one non-chain amoebot $A^* \in S'$ at a vertex of P with a strictly positive external angle, which by definition of a chain node must have a neighborhood in $S' \cup \{C\}$ that induces a connected subgraph. One can show via exhaustive search that any vertex of P with a strictly positive external angle must correspond to a node with degree at most five in the embedded G_{FCC} lattice.[5] Thus, A^* must satisfy Rule 2.

Therefore, there always exists an amoebot A^* in S satisfying an erosion rule. When A^* erodes, $S - A^*$ remains connected and contractible by Lemma 6.1, so the lemma follows by induction. □

The safety and progress lemmas yield our main theorem.

Theorem 6.3. *Assuming 3D geometric space, assorted orientations, constant-size memory, and an unfair sequential adversary, Algorithm 1 solves the leader election problem for any connected and contractible system of n amoebots in $O(n)$ rounds.*

Proof. We analyze the erosion process by applying Lemmas 6.1 and 6.2 to the structure of both null and real candidates A with $A.\text{cand} \in \{\text{NULL}, \text{TRUE}\}$. At initialization, every amoebot is a null candidate, so the structure of candidates is connected and contractible by supposition. Lemma 6.2 guarantees that some candidate satisfies an erosion rule, but this does not immediately imply it can erode: it might only be a null candidate that has not yet executed Setup, or it might be a real candidate that has null candidate neighbors. In either case, the Erode action is disabled.

Suppose that erosion progress is blocked, i.e., every candidate that satisfies an erosion rule—of which there must be at least one by Lemma 6.2—has its Erode action disabled. Since the candidate structure is connected and contractible by Lemma 6.1, the DeclareLeader action is never enabled for any amoebot until the very end, when all but one amoebot has cand = FALSE. This certainly cannot occur while there are still null candidates in the system. So even the unfair adversary has no choice but to activate some null candidate A waiting to execute Setup, which is guaranteed to be continuously enabled for A from initialization to its first activation.

Thus, regardless of the unfair adversary's choice of activation, either a null candidate executes Setup to become a real candidate or a real candidate erodes by executing Erode. The guard of Erode ensures that any such erosion occurs after the candidate's neighbors have all executed their Setup actions, avoiding any discrepancies between the erosion rules dealing only with real candidate neighbors and this proof's inclusion of null candidates. Lemma 6.1 thus guarantees that any erosion maintains the connectivity and contractibility of the candidate structure, and Lemma 6.2 guarantees once again that some candidate in the remaining structure satisfies an erosion rule. Repeatedly applying this argument shows that the candidate structure is eventually reduced to a single candidate, for which DeclareLeader becomes enabled. This is the only enabled amoebot remaining in the system, so the unfair adversary must activate it, at which point the amoebot declares itself leader and the algorithm terminates.

It remains to bound the worst-case runtime of our algorithm on a system of n amoebots, i.e., the largest number of rounds that can complete before termination based on any possible activation sequence of the unfair adversary. As observed earlier, the Setup action is continuously enabled for each amoebot from initialization to its first activation, implying that the first round does not complete until all n amoebots have executed their Setup actions, becoming (real) candidates. In the worst case, these are the only action executions that occur in the first round. At the start of the next round, Lemma 6.2 ensures there exists a candidate that satisfies an erosion rule; in the worst case, there is only one such candidate A. Because all other amoebots have moved past null candidacy, we are guaranteed that Erode is enabled for A. Thus, A must be activated and will erode by the end of the second round. More generally, it can take at most $n - 1$ rounds for $n - 1$ candidates to erode and at most one additional round for the final candidate to declare itself leader. Therefore, our algorithm terminates within $O(n)$ rounds in the worst case. □

Recall from Section 3 that G_Δ used in 2D geometric space can be treated as a single triangular lattice in G_{FCC} used in 3D geometric space. Certainly any connected and contractible 2D amoebot system is thus also a connected and contractible 3D system, yielding the following corollary.

Corollary 6.4. *Theorem 6.3 also holds in 2D geometric space.*

Notably, the special case involving erosion in squares of candidates never occurs in 2D geometric space, so an even simpler version of our algorithm that omits nbrcand variables and Rule 3 could be used to solve leader election in 2D.

7 ASYNCHRONOUS LEADER ELECTION

The *concurrency control framework* for amoebot algorithms uses the canonical amoebot model's Lock operation to provide isolation among concurrent amoebot's actions [8]. Specifically, the framework takes as input any amoebot algorithm that achieves a desired behavior under a sequential adversary and—provided the algorithm satisfies certain conventions—constructs another algorithm that produces equivalent behavior under an asynchronous adversary by carefully wrapping the original algorithm's actions in Lock calls and reorganizing their operations. In this section, we demonstrate that our algorithm for leader election by erosion is compatible with the concurrency control framework, proving the existence of an amoebot algorithm that solves leader election under an unfair asynchronous adversary. In order to apply the framework, we must first show that our algorithm satisfies the framework's three conventions:

(1) *Validity.* For all system configurations in which an action α is enabled for an amoebot A, the execution of α by A should be successful when all other amoebots are inactive.

[5] We verified this statement using a simple Mathematica program that enumerates all neighborhoods of six to eleven neighbors yielding a polyhedron vertex and verifies that its external angle is nonpositive; see the link at the end of this paper.

(2) *Phase Structure.* Each action should structure its operations as: (1) a "compute phase", during which an amoebot performs a finite amount of computation and a finite sequence of Connected, Read, and Write operations, and (2) a "move phase", during which an amoebot performs at most one movement operation decided upon in the compute phase.
(3) *Monotonicity.* Each action should be "monotonic", a lengthy technical condition describing actions' resilience to concurrent movements. Because our algorithm is *stationary*, not involving any movement operations, we omit this definition.

Lemma 7.1. *Algorithm 1 satisfies the validity, phase structure, and monotonicity conventions of the concurrency control framework for amoebot algorithms.*

Proof. Actions in Algorithm 1 only use Connected, Read, and Write operations. The validity convention requires that every isolated execution of an enabled action succeeds. Connected operations never fail. Read and Write operations only fail when the neighbor whose memory is being accessed disconnects from the calling amoebot due to a movement, which never happens in our algorithm because it is stationary. Thus, all executions of enabled actions in our algorithm succeed, satisfying validity. Phase structure can easily be verified by observing that—since our algorithm is stationary—its actions are entirely composed of compute phases. Finally, as observed in the concurrency control framework publication [8], all stationary algorithms are trivially monotonic. □

Theorem 7 of [8] states that if an algorithm satisfies the framework's three conventions and every sequential execution of the algorithm terminates, then every asynchronous execution of the corresponding algorithm produced by the framework terminates in a configuration that was also reachable by the original algorithm. Lemma 7.1 shows that our algorithm for leader election satisfies the framework's three conventions, and Theorem 6.3 shows that every sequential execution of our algorithm terminates. Therefore, we have the following corollary.

Corollary 7.2. *There exists an algorithm that solves the leader election problem for any connected and contractible amoebot system under the assumptions of 2D or 3D geometric space, assorted orientations, constant-size memory, and an unfair asynchronous adversary.*

We note that the runtime of this asynchronous algorithm depends on the overhead of the concurrency control framework which has not yet been analyzed.

8 CONCLUSION

In this work, we presented the 3D geometric space variant for the amoebot model, extending the well-established model of programmable matter to 3D space. We then presented a deterministic distributed algorithm for electing exactly one leader in $O(n)$ rounds under an unfair sequential adversary (Theorem 6.3) for connected and contractible systems, extending and simplifying the erosion-based approach of Di Luna et al. [11]. This algorithm can be flexibly applied to both 2D and 3D space (Corollary 6.4) and improves over the related algorithm for 3D leader election by Gastineau et al. [17] by extending the set of amoebot orientations and replacing the assumption of 2-neighborhood vision with local comparisons between 1-neighbors' orientations. Finally, we proved that this algorithm can be transformed using the concurrency control framework for amoebot algorithms [8] to obtain the first known amoebot algorithm that solves leader election under an unfair asynchronous adversary (Corollary 7.2).

Although our leader election algorithm is the first to make use of the concurrency control framework for asynchronous correctness, several others discussed in Section 4 could likely do the same. The proof of Lemma 7.1 suggests that any stationary algorithm adhering to the standard amoebot assumptions (e.g., 1-neighborhood vision) will likely satisfy the framework's three conventions, and all of the deterministic algorithms already have proofs of termination under a sequential adversary or some weaker scheduler. Some work is required to translate existing algorithms into the action-based semantics used by the canonical amoebot model and to reprove their correctness with respect to the new formulation, but the framework's subsequent application seems straightforward.

One potentially interesting extension of this work is to modify the leader election algorithm such that when it is run on an initially connected amoebot system that *contains holes*, the existence of holes can be identified in a distributed way. Just as the topological definition of holes identifies the inability to shrink the object to a single point, so too could the inability for our algorithm to erode a system to a single amoebot be a clue about the existence of holes. Our algorithm already confirms the absence of holes as this is a necessary condition for a leader to emerge, but perhaps could be extended—likely with additional amoebot communication—to decide when holes exist.

Finally, we showed that our algorithm for leader election in 3D geometric space could be immediately applied without modification to a 2D geometric system as a special case (Corollary 6.4). This may not necessarily be true of every problem and algorithm. What existing algorithms for 2D geometric systems can be viewed as special cases of generalized 3D algorithms? What fundamental characteristics of a given algorithm or problem inhibit this translation? Answers to these questions will accelerate the development of 3D amoebot algorithms and their potential applications to amoebots' modular robotic counterparts.

SOURCE CODE AVAILABILITY

All Mathematica source code used to generate this paper's figures and verify the claims in the proofs of Lemmas 6.1 and 6.2 is openly available at https://github.com/SOPSLab/3DLeaderElection-Mathematica.

ACKNOWLEDGMENTS

J.L.B. and A.W.R. are supported in part by the National Science Foundation under awards CCF-1733680 and CCF-2106917 and by the U.S. Army Research Office under award MURI W911NF-19-1-0233. J.J.D. is supported by the Momental Foundation under the Mistletoe Research Fellowship and by the ASU Biodesign Institute.

REFERENCES

[1] Rida A. Bazzi and Joseph L. Briones. 2019. Stationary and Deterministic Leader Election in Self-Organizing Particle Systems. In *Stabilization, Safety, and Security of Distributed Systems (Lecture Notes in Computer Science, Vol. 11914)*. 22–37. https://doi.org/10.1007/978-3-030-34992-9_3

[2] Gregory S. Chirikjian. 1994. Kinematics of a Metamorphic Robotic System. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. 449–455. https://doi.org/10.1109/ROBOT.1994.351256

[3] Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schröder. 2013. Digital Geometry Processing with Discrete Exterior Calculus. In *ACM SIGGRAPH 2013 Courses (SIGGRAPH '13)*. 7:1–7:126. https://doi.org/10.1145/2504435.2504442

[4] Gianlorenzo D'Angelo, Mattia D'Emidio, Shantanu Das, Alfredo Navarra, and Giuseppe Prencipe. 2020. Asynchronous Silent Programmable Matter Achieves Leader Election and Compaction. *IEEE Access* 8 (2020), 207619–207634. https://doi.org/10.1109/ACCESS.2020.3038174

[5] Joshua J. Daymude. 2021. *Collaborating in Motion: Distributed and Stochastic Algorithms for Emergent Behavior in Programmable Matter*. Ph. D. Dissertation. Arizona State University, Tempe, AZ. http://login.ezproxy1.lib.asu.edu/login?url=https://www.proquest.com/dissertations-theses/collaborating-motion-distributed-stochastic/docview/2531565717/se-2?accountid=4485.

[6] Joshua J. Daymude, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. 2017. Improved Leader Election for Self-Organizing Programmable Matter. In *Algorithms for Sensor Systems (Lecture Notes in Computer Science, Vol. 10718)*. 127–140. https://doi.org/10.1007/978-3-319-72751-6_10

[7] Joshua J. Daymude, Kristian Hinnenthal, Andréa W. Richa, and Christian Scheideler. 2019. Computing by Programmable Particles. In *Distributed Computing by Mobile Entities*. Lecture Notes in Computer Science, Vol. 11340. Springer, Cham, 615–681. https://doi.org/10.1007/978-3-030-11072-7_22

[8] Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. 2021. The Canonical Amoebot Model: Algorithms and Concurrency Control. In *35th International Symposium on Distributed Computing (DISC 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 209)*. 20:1–20:19. https://doi.org/10.4230/LIPIcs.DISC.2021.20

[9] Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. 2014. Amoebot - a New Model for Programmable Matter. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*. 220–222. https://doi.org/10.1145/2612669.2612712

[10] Zahra Derakhshandeh, Robert Gmyr, Thim Strothmann, Rida Bazzi, Andréa W. Richa, and Christian Scheideler. 2015. Leader Election and Shape Formation with Self-Organizing Programmable Matter. In *DNA Computing and Molecular Programming (Lecture Notes in Computer Science, Vol. 9211)*. 117–132. https://doi.org/10.1007/978-3-319-21999-8_8

[11] Giuseppe A. Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. 2020. Shape Formation by Programmable Particles. *Distributed Computing* 33, 1 (2020), 69–101. https://doi.org/10.1007/s00446-019-00350-6

[12] Fabien Dufoulon, Shay Kutten, and William K. Moses Jr. 2021. Efficient Deterministic Leader Election for Programmable Matter. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. 103–113. https://doi.org/10.1145/3465084.3467900

[13] Yuval Emek, Shay Kutten, Ron Lavi, and William K. Moses Jr. 2019. Deterministic Leader Election in Programmable Matter. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019) (Leibniz International Proceedings in Informatics (LIPIcs))*. 140:1–140:14. https://doi.org/10.4230/LIPICS.ICALP.2019.140

[14] Robert Fitch and Zack Butler. 2008. Million Module March: Scalable Locomotion for Large Self-Reconfiguring Robots. *The International Journal of Robotics Research* 27, 3-4 (2008), 331–343. https://doi.org/10.1177/0278364907085097

[15] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro (Eds.). 2019. *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*. Lecture Notes in Computer Science, Vol. 11340. Springer, Cham. https://doi.org/10.1007/978-3-030-11072-7

[16] Nicolas Gastineau, Wahabou Abdou, Nader Mbarek, and Olivier Togni. 2019. Distributed Leader Election and Computation of Local Identifiers for Programmable Matter. In *Algorithms for Sensor Systems (Lecture Notes in Computer Science, Vol. 11410)*. 159–179. https://doi.org/10.1007/978-3-030-14094-6_11

[17] Nicolas Gastineau, Wahabou Abdou, Nader Mbarek, and Olivier Togni. 2022. Leader Election and Local Identifiers for Three-dimensional Programmable Matter. *Concurrency and Computation: Practice and Experience* 34, 7 (2022), e6067. https://doi.org/10.1002/cpe.6067

[18] Guanqi Liang, Haobo Luo, Ming Li, Huihuan Qian, and Tin Lun Lam. 2020. FreeBOT: A Freeform Modular Self-Reconfigurable Robot with Arbitrary Connection Point - Design and Implementation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 6506–6513. https://doi.org/10.1109/IROS45743.2020.9341129

[19] Haobo Luo and Tin Lun Lam. 2022. Adaptive Flow Planning of Modular Spherical Robot Considering Static Gravity Stability. *IEEE Robotics and Automation Letters* 7, 2 (2022), 4228–4235. https://doi.org/10.1109/LRA.2022.3150028

[20] Haobo Luo, Ming Li, Guangqi Liang, Huihuan Qian, and Tin Lun Lam. 2020. An Obstacle-Crossing Strategy Based on the Fast Self-Reconfiguration for Modular Sphere Robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 3296–3303. https://doi.org/10.1109/IROS45743.2020.9341162

[21] Othon Michail and Paul G. Spirakis. 2016. Simple and Efficient Local Codes for Distributed Stable Network Construction. *Distributed Computing* 29, 3 (2016), 207–237. https://doi.org/10.1007/s00446-015-0257-4

[22] Matthew J. Patitz. 2014. An Introduction to Tile-Based Self-Assembly and a Survey of Recent Results. *Natural Computing* 13, 2 (2014), 195–224. https://doi.org/10.1007/s11047-013-9379-4

[23] Benoit Piranda and Julien Bourgeois. 2018. Designing a Quasi-Spherical Module for a Huge Modular Robot to Create Programmable Matter. *Autonomous Robots* 42 (2018), 1619–1633. https://doi.org/10.1007/s10514-018-9710-0

[24] John W. Romanishin, Kyle Gilpin, Sebastian Claici, and Daniela Rus. 2015. 3D M-Blocks: Self-Reconfiguring Robots Capable of Locomotion via Pivoting in Three Dimensions. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 1925–1932. https://doi.org/10.1109/ICRA.2015.7139450

[25] Petras Swissler and Michael Rubenstein. 2020. FireAnt3D: A 3D Self-Climbing Robot towards Non-Latticed Robotic Self-Assembly. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 3340–3347. https://doi.org/10.1109/IROS45743.2020.9341116

[26] Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. 2021. Engineering Efficient and Massively Parallel 3D Self-Reconfiguration Using Sandboxing, Scaffolding and Coating. *Robotics and Autonomous Systems* 146 (2021), 103875. https://doi.org/10.1016/j.robot.2021.103875

[27] Tommaso Toffoli and Norman Margolus. 1991. Programmable Matter: Concepts and Realization. *Physica D: Nonlinear Phenomena* 47, 1-2 (1991), 263–272. https://doi.org/10.1016/0167-2789(91)90296-L

[28] Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. 2013. Active Self-Assembly of Algorithmic Shapes and Patterns in Polylogarithmic Time. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*. 353–354. https://doi.org/10.1145/2422436.2422476

[29] Ryonosuke Yamada and Yukiko Yamauchi. 2022. Search by a Metamorphic Robotic System in a Finite 3D Cubic Grid. In *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 221)*. 20:1–20:16. https://doi.org/10.4230/LIPIcs.SAND.2022.20

[30] Yukiko Yamauchi, Taichi Uehara, Shuji Kijima, and Masafumi Yamashita. 2017. Plane Formation by Synchronous Mobile Robots in the Three-Dimensional Euclidean Space. *J. ACM* 64, 3 (2017), 1–43. https://doi.org/10.1145/3060272

[31] Mark Yim, Ying Zhang, John Lamping, and Eric Mao. 2001. Distributed Control for 3D Metamorphosis. *Autonomous Robots* 10, 1 (2001), 41–56. https://doi.org/10.1023/A:1026544419097