$\mathcal{O}$-Hull Formation for Programmable Matter

Joshua J. Daymude[1], Robert Gmyr[2], Kristian Hinnenthal[3], Irina Kostitsyna[4], Christian Scheideler[3], and Andréa W. Richa[1]

1 **Computer Science, CIDSE, Arizona State University, Tempe, AZ, USA**
{jdaymude, aricha}@asu.edu
2 **Department of Computer Science, University of Houston, Houston, TX, USA**
rgmyr@uh.edu
3 **Department of Computer Science, Paderborn University, Paderborn, Germany**
{krijan, scheidel}@mail.upb.de
4 **Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands**
i.kostitsyna@tue.nl

1 Introduction

Research in *self-organizing programmable matter* is becoming increasingly popular in many fields with potential for broad applications, for example, in nanomedicine. Imagine tiny particles locating and repairing small wounds in the human body, or capturing harmful cells and transporting them out of the body. In this context various problems such as *shape formation* [6, 15, 10, 12], *coating* [7, 1], and *shape recognition* [9] have recently been investigated under various theoretical models. Somewhat in between these lies the problem of *shape sealing*, where the goal is to isolate an object by enclosing it with a shell of particles.

In this paper we study the problem of sealing a 2D object under the *amoebot model* [5, 4], which models programmable matter as a collection of nanoscale agents (called *particles*) with limited computational capabilities that move on a grid and can locally exchange information in order to collectively achieve a given goal.

The Amoebot Model. In the amoebot model the underlying geometry is an infinite triangular lattice $G_\triangle = (V, E)$. Each particle occupies either a single node in V (*contracted* particle) or a pair of adjacent nodes in V (*expanded* particle). Particles move via a series of *expansions* and *contractions*: a contracted particle can expand into an unoccupied adjacent node, and contract into one of its nodes (see Fig. 1). Neighboring particles can coordinate their movements in a *handover*, which can occur when: a contracted particle P "pushes" an expanded neighbor Q by expanding into a node occupied by Q, forcing it to contract; or an expanded particle Q "pulls" a contracted neighbor P by contracting, forcing P to expand into the node it is vacating. Handovers help maintain the connectivity of the particle system.

The particles are assumed to be anonymous, with no global coordinate system or compass. The only assumption is that the particles have a common *chirality*, which allows them to number the incident edges in clockwise order.

We assume the standard asynchronous model of distributed computing (see, e.g., [11]). A classical result under this model states that for any concurrent asynchronous execution of atomic actions, there exists a sequential ordering of actions producing the same end result, provided conflicts that arise in the concurrent execution are resolved. In the amoebot model, an atomic action corresponds to a single particle activation in which a particle can perform some computation involving its memory and the memories of its neighbors and at most one expansion or contraction. Conflicts involving concurrent memory writes or simultaneous particle expansions into the same unoccupied node are resolved arbitrarily such that at most one particle is writing into a given memory location or expanding into a given node at a time.

35th European Workshop on Computational Geometry, Utrecht, The Netherlands, March 18–20, 2019.
This is an extended abstract of a presentation given at EuroCG'19. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

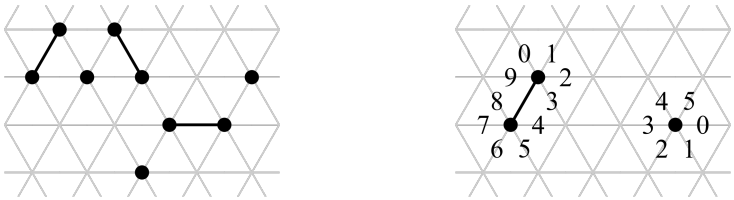

Figure 1 Left: Expanded and contracted particles. Right: particles with edge numbering.

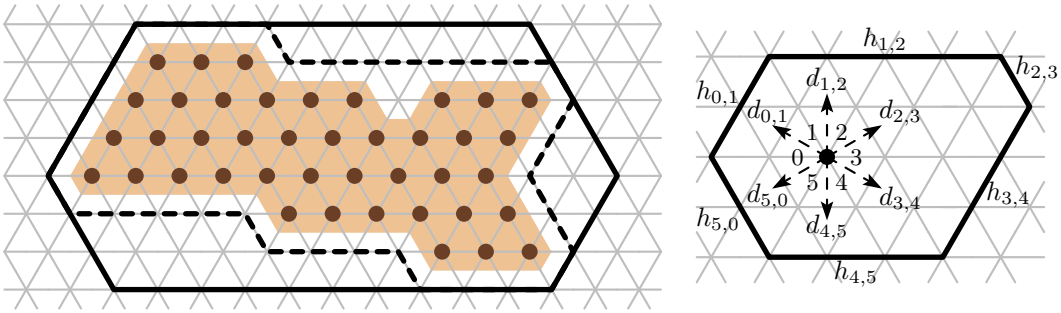

Figure 2 Left: An example of an object S (highlighted in orange), enclosed by its strong $\mathcal{O}$-hull (solid line) and its $\mathcal{O}$-hull (dashed line). Right: A particle's local labeling of the six half-planes composing the strong $\mathcal{O}$-hull; $\mathcal{D} = \{2, 2, 4, 3, 2, 2\}$.

While in reality many particles may be active concurrently, when analyzing our algorithms it suffices to consider a sequence of activations where only one particle is active at a time. We assume the activation sequence is *fair*: any particle P will be activated at some future time. An *asynchronous round* is complete once all particles have been activated at least once.

Problem Description. Shape sealing in two dimensions reduces to enclosing an object in a cycle. To optimize the number of particles needed to seal an object, we study the problem of particles forming a convex hull. The amoebot model limits the movement of the particles to three directions, thus we build a *restricted-orientation hull*, or an *$\mathcal{O}$-hull*, of a given object.

The notions of $\mathcal{O}$-convexity and an $\mathcal{O}$-hull were introduced by Rawlins [14] (see also [8]). Given a set of fixed orientations $\mathcal{O}$, a set is *$\mathcal{O}$-convex* if its intersection with any line with one of the orientations in $\mathcal{O}$ is connected. An *$\mathcal{O}$-hull* of a given set S is defined as an intersection of all $\mathcal{O}$-convex sets containing S. Furthermore, a *strong $\mathcal{O}$-hull* of S is an intersection of all half-planes bounded by lines with orientations in $\mathcal{O}$ and containing S. In our case $\mathcal{O}$ consists of three orientations of the axes of the triangular grid $G_\triangle$.

Let S be a simply-connected subgraph of $G_\triangle$, and $\mathcal{P}$ be a connected system of initially contracted amoebot particles on $G_\triangle$ (non-overlapping with S). The shape sealing problem is to reconfigure $\mathcal{P}$ within $G_\triangle \setminus S$ so that every node of the $\mathcal{O}$-hull of S is occupied by a contracted particle. Note that as the particles are not allowed to occupy the nodes of S, the hull that will be constructed by $\mathcal{P}$ will in fact be the offset-by-one $\mathcal{O}$-hull of S (see Fig. 2 (left)). Nevertheless, to simplify the exposition, we will refer to it using the same term $\mathcal{O}$-hull. We further assume that $\mathcal{P}$ has enough particles to form an $\mathcal{O}$-hull, that it contains a unique leader particle[1] ℓ initially adjacent to S, and that S does not contain any *tunnels* of width 1

[1] Such a particle can be determined in $O(|\mathcal{P}|)$ asynchronous rounds with high probability using a slightly modified version of the leader election algorithm of [3].

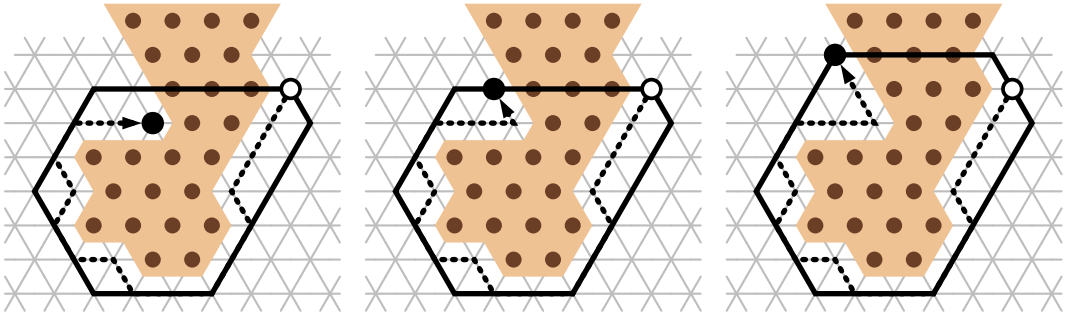

Figure 3 A particle (black dot) estimates strong $\mathcal{O}$-hull (black) after having traversed the dotted path from its starting point (black circle). Left: $d_h \geq 1$ for all $h \in \mathcal{H}$, the next move does not push any half-plane. Middle: $d_{1,2} = 0$. Right: Half-plane $d_{1,2}$ has been pushed.

(that is, $G_\triangle \setminus S$ is 2-connected).

We present a fully distributed, local algorithm for the shape sealing problem that runs in $O(B+H)$ asynchronous rounds, where B is the perimeter of S, and H is the perimeter of the $\mathcal{O}$-hull of S. We present the algorithm in three parts: we first describe how a single particle with unbounded memory can find the strong $\mathcal{O}$-hull using a simple geometric observation. The main difficulty of our result lies in emulating the single-particle algorithm with a system of bounded-memory particles. The final part, converting the strong $\mathcal{O}$-hull into the $\mathcal{O}$-hull, is rather straightforward: all particles at the convex vertices that are not adjacent to S can move inside thus "deflating" the strong $\mathcal{O}$-hull towards the $\mathcal{O}$-hull; after additional $O(H)$ asynchronous rounds the $\mathcal{O}$-hull is achieved. In the rest of this extended abstract we present the first two parts of the algorithm. Due to space constraints, we omit the proofs of the theorems, which can be found in the full version of this paper [2].

2 Single Particle Algorithm

Consider a single particle P with unbounded memory. Let P be initially placed somewhere adjacent to S. The main idea of this algorithm is to let P traverse the boundary of S clockwise, while internally maintaining a representation of the strong $\mathcal{O}$-hull. The strong $\mathcal{O}$-hull can be represented with six half-planes $\mathcal{H} = \{h_{0,1}, h_{1,2}, \ldots, h_{5,0}\}$, which P can label clockwise (in Fig. 2 (right)). Particle P computes the locations of these half-planes by maintaining six counters $\mathcal{D} = \{d_h : h \in \mathcal{H}\}$, where d_h holds the distance from P to the line supporting h. If one of these counters is 0, P is on the current estimate of the strong $\mathcal{O}$-hull.

Counters initially are set to 0. As P moves, it always stays parallel to two half-planes; their counters do not get updated. For two other half-planes P gets further away; their counters get incremented. For each of the remaining two half-planes, P either moves closer to it (if its counter was > 0) or steps outside of the half-plane (if the counter was $= 0$). In the former case the counter gets decremented, in the latter case the counter does not get updated which corresponds to a half-plane getting "pushed". Refer to Fig. 3 for an example.

Finally, P needs to detect when it has computed the strong $\mathcal{O}$-hull. To do so, it stores six terminating bits $\{b_h : h \in \mathcal{H}\}$, where $b_h = 1$ if P has visited the line supporting half-plane h since it last pushed any half-plane, and $b_h = 0$ otherwise. Whenever P moves without pushing a half-plane, for each h with $d_h = 0$ it sets b_h to 1. Otherwise, when P pushes a half-plane, it sets b_h to 0 for all h. If after a move all six terminating bits are 1, P contracts and terminates.

▶ **Theorem 2.1.** *The single-particle algorithm terminates after $O(B)$ asynchronous rounds with particle P holding the correct representation of the strong $\mathcal{O}$-hull in the six counters $\mathcal{D}$.*

3 The Strong $\mathcal{O}$-hull Algorithm

Next we show how a system of n particles each with only constant-size memory can emulate the single-particle algorithm. The leader particle ℓ of $\mathcal{P}$ is primarily responsible for emulating the particle with unbounded memory in the single-particle algorithm. To do so, it utilizes the other particles in the system as distributed memory. More precisely, as ℓ moves, it will create a chain of particles behind it that will be used to store the distances d_h from ℓ to the lines supporting half-planes h as binary counters. Once these measurements are complete, ℓ uses them to lead the other particles in forming the $\mathcal{O}$-hull.

A Binary Counter of Particles. We build upon an increment-only binary counter under the amoebot model [13]. Suppose that the participating particles are organized as a simple chain with the leader at its front. Each particle P has a bit value $P.bit$, that can be empty if P is not part of the counter. A *final token* f is held by a particle marking the end of the counter. Then the counter value is represented by the bits of the particles from ℓ (holding the least significant bit) up to the particle holding the token f.

The leader ℓ initiates counter operations, and the rest of the particles carry these operations out. To increment (decrement) the counter, the leader ℓ generates an increment token c^+ (decrement token c^-). The tokens are consumed or passed along the chain (as a carry bit) while updating the bits of the particles accordingly until they are consumed. To test whether the counter is 0, the leader checks the status of its follower counter particle P_1. If P_1 is holding a decrement token c^- and $P_1.bit = 1$, ℓ cannot conclusively test whether the counter's value is 0. Otherwise, the counter value is 0 if and only if $\ell.bit = 0$, P_1 is holding the final token f, and P_1 is not holding an increment token c^+.

The proof of the following theorem is rather involved, we omit it due to space constraints.

▶ **Theorem 3.1.** *Given any fair asynchronous activation sequence of the particles, and any nonnegative sequence of m operations, the distributed binary counter correctly processes all operations in $O(m)$ asynchronous rounds.*

3.1 Estimating the Strong $\mathcal{O}$-Hull

We now combine the movement rules of the single-particle algorithm with our distributed, multi-particle binary counter to enable the leader to compute the strong $\mathcal{O}$-hull of S.

First, using the *spanning tree primitive* (see, e.g., [7]), a spanning tree is constructed on the particle system rooted at the leader ℓ. Each activated particle P, if it has a neighbor Q already in the tree, becomes a *follower* and sets $P.parent$ to Q. Note that the leader can immediately begin estimating the strong $\mathcal{O}$-hull without waiting for the entire tree to form.

The first few followers of ℓ form a counter chain and store six counters d_h in a distributed fashion. Imitating the single-particle algorithm, ℓ performs a clockwise traversal of the boundary of S using the right-hand rule, updating its counters along the way. It terminates once it has moved in all six directions without pushing a half-plane, which it detects using its six terminating bits b_h. In the multi-particle setting, we need to carefully consider both how ℓ interacts with its followers as it moves and how it updates its counters.

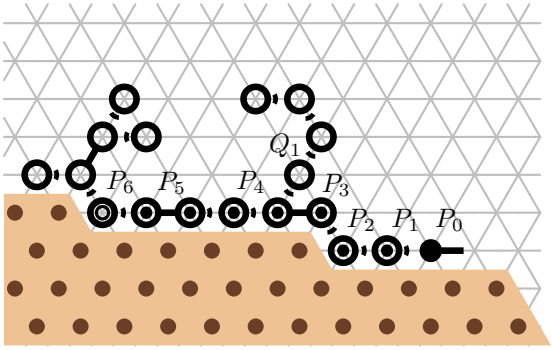

Figure 4 The leader P_0 and its followers. Followers with dots are on the counter, and P_6 holds the final token. Particle Q_1 cannot handover with P_3, while all other potential handovers are safe.

Rules for Distributed Counters. The increment and decrement tokens are handled as described above. However, as at the beginning the particles form a tree, and not a simple path, we enforce that the counters are only extended along followers on the object's boundary.

As there may be role-swaps of the leader ℓ (described below), to maintain connectivity of the counters we modify them to store two bits per particle. Then if a role-swap occurs, the counter bits will need to be shifted towards the leader to keep all the bits of the particles closest to ℓ "full". This can be easily achieved by passing the bit value when a counter particle P detects that there is a bit value missing in the memory of its parent $P.parent$.

Rules for Leader Computation and Movement. First suppose ℓ is contracted. If all its terminating bits $b_h = 1$, then ℓ has computed the strong $\mathcal{O}$-hull. Else, if the zero-test operation is unavailable on any of the counters, ℓ skips its turn; otherwise, ℓ will attempt to expand into the node v along the boundary of S. If v is unoccupied, or is occupied by an expanded particle, ℓ calculates the updated distances $\mathcal{D}$, generates the corresponding increment/decrement tokens and expands into v. If v is occupied by a contracted particle P, ℓ will have to initiate a role-swap with P, such that P becomes the new leader and ℓ becomes its follower (the second particle in the counter). This is allowed only if ℓ has its both bit values full, or if it is holding a final token f_h. In the former case ℓ passes the value only of one least significant bit to P (this is where the two bits are used to maintain the connectivity), and in the latter case ℓ passes a bit (if it exists) and the final token to P. It also updates its terminating bits b_h for all $h \in \mathcal{H}$.

Finally, if ℓ is expanded, let P be its follower child emulating bits of the counters. Then if P is contracted, ℓ pulls P in a handover.

Rules for Follower Movement. For any follower P, if it is expanded and has no children in the spanning tree nor any non-tree neighbor, then it simply contracts. If P is contracted and is following the tail of its expanded parent $Q = P.parent$, then P pushes Q in a handover. Similarly, if P is expanded and has a contracted child Q, P pulls Q in a handover. However, we do not allow handovers that may disconnect the counters (see Fig. 4).

Once the counters contain an accurate representation of the strong $\mathcal{O}$-hull, the leader ℓ can simply lead the particle system in tracing it out by traversing the strong $\mathcal{O}$-hull in clockwise order. While moving along the strong $\mathcal{O}$-hull, ℓ uses its distributed counters to detect when it reaches a vertex of the strong $\mathcal{O}$-hull, at which point it turns 60° to follow the next half-plane, and so on. The movement rules for the leader and the followers in this

phase are very similar to those of the previous phase. With some careful analysis we can show the following theorem.

▶ **Theorem 3.2.** *The presented algorithm solves the strong $\mathcal{O}$-hull formation problem for an object S in $O(B + H)$ asynchronous rounds in the worst case.*

Acknowledgments. Joshua J. Daymude and Andrá W. Richa are supported in part by NSF Awards CCF-1637393 and CCF-1733680.

References

1 J. J. Daymude, Z. Derakhshandeh, R. Gmyr, A. Porter, A. W. Richa, C. Scheideler, and T. Strothmann. On the runtime of universal coating for programmable matter. *Natural Computing*, 17(1):81–96, 2018.
2 J. J. Daymude, R. Gmyr, K. Hinnenthal, I. Kostitsyna, C. Scheideler, and A. W. Richa. Convex hull formation for programmable matter. Available on arXiv, 2018.
3 J. J. Daymude, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Improved leader election for self-organizing programmable matter. In *Algorithms for Sensor Systems (ALGOSENSORS)*, pages 127–140, 2017.
4 J. J. Daymude, A. W. Richa, and C. Scheideler. The amoebot model. Available online at `https://sops.engineering.asu.edu/sops/amoebot`, 2017.
5 Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Brief announcement: Amoebot - a new model for programmable matter. In *Proc. 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 220–222, 2014.
6 Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal shape formation for programmable matter. In *Proc. 28th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–299, 2016.
7 Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal coating for programmable matter. *Theoretical Computer Science*, 671:56–68, 2017.
8 E. Fink and D. Wood. *Restricted-Orientation Convexity*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag Berlin Heidelberg, 2004.
9 R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, and C. Scheideler. Shape Recognition by a Finite Automaton Robot. In *Proc. 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 52:1–52:15, 2018.
10 F. Hurtado, E. Molina, S. Ramaswami, and V. Sacristán. Distributed reconfiguration of 2D lattice-based modular robotic systems. *Autonomous Robots*, 38(4):383–413, 2015.
11 N. Lynch. *Distributed Algorithms*. Morgan Kauffman, 1996.
12 M. J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.
13 A. Porter and A. W. Richa. Collaborative computation in self-organizing particle systems. In *Proc. 17th International Conference on Unconventional Computing and Natural Computation (UCNC)*, 2018.
14 G. J. E. Rawlins. *Explorations in Restricted Orientation Geometry*. PhD thesis, University of Waterloo, 1987. AAI0561642.
15 D. Woods, H.-L. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proc. 4th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 353–354, 2013.