Universal Shape Formation for Programmable Matter

Zahra Derakhshandeh
Arizona State University
zderakhs@asu.edu

Robert Gmyr
Paderborn University
gmyr@mail.upb.de

Andréa W. Richa
Arizona State University
aricha@asu.edu

Christian Scheideler
Paderborn University
scheideler@upb.de

Thim Strothmann
Paderborn University
thim@mail.upb.de

ABSTRACT

We envision programmable matter consisting of systems of computationally limited devices (which we call *particles*) that are able to self-organize in order to achieve a desired collective goal without the need for central control or external intervention. Central problems for these particle systems are shape formation and coating problems. In this paper, we present a *universal* shape formation algorithm which takes an arbitrary shape composed of a constant number of equilateral triangles of unit size and lets the particles build that shape at a scale depending on the number of particles in the system. Our algorithm runs in $O(\sqrt{n})$ asynchronous execution rounds, where n is the number of particles in the system, provided we start from a well-initialized configuration of the particles. This is optimal in a sense that for any shape deviating from the initial configuration, any movement strategy would require $\Omega(\sqrt{n})$ rounds in the worst case (over all asynchronous activations of the particles). Our algorithm relies only on local information (e.g., particles do not have ids, nor do they know n, or have any sort of global coordinate system), and requires only a constant-size memory per particle.

Keywords

Self-Organizing Systems; Programmable Matter; Distributed Algorithms

1. INTRODUCTION

The vision behind *programmable matter* is to have a piece of matter that can change its physical properties like shape, density, conductivity, or color in a programmable fashion based on either user input or autonomous sensing. Many realizations of programmable matter have been proposed — ranging from DNA tiles, shape-changing molecules, and synthetic cells, to reconfigurable modular robotics — each pursuing solutions for specific application scenarios with their own, special capabilities and constraints. We envision programmable matter to consist of a system of computationally limited devices (which we refer to as particles) which can move and bond and exchange information in order to solve the given goal in a collective manner and without any outside intervention. One can envision a number of applications for these particle systems, most prominently shape formation and coating problems. Various algorithms for the formation of specific shapes have already been proposed within our model, but they are all inherently sequential: starting with a seed particle, the particles bond to each other in a chain-like fashion till the desired shape has been constructed. So for n particles $\Omega(n)$ rounds are needed to finish the shape formation. In this work we do not only show how to design a universal shape formation algorithm but we also show that for any shape composed of a constant number of equilateral triangles of unit size, our approach just needs $O(\sqrt{n})$ rounds to arrive at the desired shape, when starting in a well-initialized shape. We also show that this bound is best possible. Our shape formation approach is based on the geometric amoebot model [11], which we briefly describe in the next section.

1.1 Amoebot model

We assume that any structure the particle system can form can be represented as a subgraph of an infinite graph $G = (V, E)$ where V represents all possible positions the particles can occupy relative to their structure, and E represents all possible atomic transitions a particle can perform as well as all places where neighboring particles can bond to each other. In the *geometric amoebot model*, we assume that $G = G_{\text{eqt}}$, where G_{eqt} is the infinite regular triangular grid graph. Figure 1(a) illustrates the standard planar embedding of G_{eqt}.

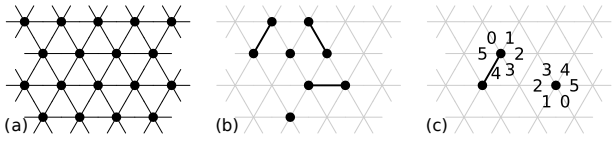

Figure 1: (a) shows a section of G_{eqt}. Nodes of G_{eqt} are shown as black circles. (b) shows five particles on G_{eqt}. The underlying graph G_{eqt} is depicted as a gray mesh. A particle occupying a single node is depicted as a black circle, and a particle occupying two nodes is depicted as two black circles connected by an edge. (c) depicts two particles occupying two non-adjacent positions on G_{eqt}. The particles have different offsets for their head port labelings.

SPAA '16, July 11-13, 2016, Pacific Grove, CA, USA

DOI: http://dx.doi.org/10.1145/2935764.2935784

We recall the main properties of the geometric amoebot model from [11]. Each particle occupies either a single node or a pair of adjacent nodes in G_{eqt}, and every node can be occupied by at most one particle. Two particles occupying adjacent nodes are *connected* by a *bond*, and we refer to such particles as *neighbors*. The bonds do not just ensure that the particles form a connected structure but they are also used for exchanging information as explained below.

Particles move through *expansions* and *contractions*: If a particle occupies one node (i.e., it is *contracted*), it can expand to an unoccupied adjacent node to occupy two nodes. If a particle occupies two nodes (i.e., it is *expanded*), it can contract to one of these nodes to occupy only a single node. Figure 1(b) illustrates a set of particles (some contracted, some expanded) on the underlying graph G_{eqt}. For an expanded particle, we denote the node the particle last expanded into as the *head* of the particle and call the other occupied node its *tail*. For a contracted particle, the single node occupied by the particle is both its head and its tail. A *handover* allows particles to stay connected as they move. Two scenarios are possible here: (1) a contracted particle p can "push" a neighboring expanded particle q and expand into the neighboring node previously occupied by q, forcing q to contract, or (2) an expanded particle p can "pull" a neighboring contracted particle q to node v it occupies thereby causing q to expand into v, which allows p to contract.

Particles are *anonymous*. Each particle has a collection of *ports*, one for each edge incident to the nodes occupied by it, that have unique labels from the local perspective of that particle. Adjacent particles establish bonds through the ports facing each other. We assume that the particles have a common *chirality*, i.e., they all have the same notion of *clockwise direction*, which allows a particle to order the port labels of its head and tail in clockwise order. However, particles do not have a common sense of orientation since they can have different offsets of the labelings, see Figure 1(c). Without loss of generality, we assume that each particle labels its head and tail ports from 0 to 5 in clockwise order. Whenever a particle p is connected through some port to a particle q, we assume that p knows the label of q's port that lies opposite of the respective port of p. Futhermore, we assume that p knows whether q's port belongs to the head or tail of q.

Each particle has a constant-size shared local memory that can be read and written to by any neighboring particle. This allows a particle to exchange information with a neighboring particle by simply writing it into the other particle's memory. A particle always knows whether it is contracted or expanded, and in the latter case it also knows along which head port label it is expanded. We assume that this information is also available to the neighboring particles (by publishing that label in its local shared memory). Particles do not know the total number of particles, nor do they have any estimate on this number.

We assume the standard asynchronous model from distributed computing, where the particle system progresses through a sequence of *particle activations*, i.e., only one particle is active at a time. Whenever a particle is activated, it can perform an arbitrary bounded amount of computation (involving its local memory as well as the shared memories of its neighbors) and at most one movement. We define a *round* to be over once each particle has been activated at least once.

The *configuration* C of the system at the beginning of time t consists of the nodes in G_{eqt} occupied by the set of particles; in addition, for every particle p, C contains the current state of p, including whether the particle is expanded or contracted, its port labeling, and the contents of its local memory. For more details on the model, please refer to [11].

1.2 Related work

Many approaches have already been proposed that can potentially be used for shape formation. One can distinguish between active and passive systems. In passive systems the particles either do not have any intelligence at all (but just move and bond based on their structural properties or due to chemical interactions with the environment), or they have limited computational capabilities but cannot control their movements. Examples of research on shape formation in *passive systems* appear in DNA self-assembly systems (see, e.g., [9, 17, 20]), and also the surveys in [16, 23]). In particular, [17] presents an approach to fold long single-stranded DNA molecules into arbitrary 2D shapes. In [15, 14], the authors study network topology and shape construction problems under a variation of the population protocols model [1], focusing on specific simple structures, such as a spanning line, a spanning star, or a spanning square.

In *active systems*, computational particles can control the way they act and possibly move in order to solve a specific task. Robotic swarms, modular robotic systems, and cellular automata are some examples of active programmable matter systems. In *swarm robotics*, one usually has a collection of autonomous robots that have limited sensing, often including vision, and communication ranges, and that can freely move in a given area. They follow a variety of goals, including shape formation (e.g., [18, 2, 12]). For example, in [18], the authors demonstrate that programmable self-assembly of complex two-dimensional shapes with hundreds or thousands of simple robots called *kilobots* is possible; however their approach differs considerably from ours, since the algorithms rely on a global pre-processing phase for shape formation that directly depends on the number of robots in the system. In [2], the authors explore a method to build and repair arbitrary two-dimensional shapes on a 2D coordinate system with a swarm of self-assembly robots. In [12], the authors study the algorithmic limitations of building a predefined exact shape by a set of autonomous mobile robots. Samples of other representative work can be found in e.g., [19, 7, 8]. While the analytic techniques developed in the area of swarm robotics and natural swarms are of some relevance for this work, the individual units in those systems have more powerful communication and processing capabilities than the systems we consider.

Beyond conventional actuation, sensing, and control typically found in fixed-morphology robots, *self-reconfigurable robots* are also able to adapt to form desired shapes (see e.g., [25, 26, 27]). *Metamorphic robots* form a subclass of self-reconfigurable robots that share some characteristics of our geometric model [6]. The hardware development in the field of self-reconfigurable robotics has been complemented by a number of algorithmic advances (e.g., [3, 22, 18]), but so far mechanisms that scale from a few to hundreds or thousands of individual units are still under investigation. While pattern formation has been extensively studied in the *cellular automata* model (e.g., in [21, 4, 13]), these studies differ from our model since particles can replicate (or die) at will.

The *nubot* model [24, 5] aims at providing the theoretical framework that would allow for a more rigorous algorithmic study of biomolecular-inspired systems, more specifically of self-assembly systems with active molecular components. One of the main results of [24] is to efficiently construct two-dimensional geometric shapes in polylogarithmic time in the size of the shape. While bio-molecular inspired systems share many similarities with self-organizing particle systems, there are differences — e.g., the system allows for an additional (non-local) notion of rigid-body movement.

Finally, in [10], we presented a preliminary algorithmic framework in the geometric amoebot model that allows particles to to build simple shapes, such as a hexagon or a triangle. The number of rounds required by the algorithms presented in this work is at least linear on the number of particles in the worst-case.

1.3 Our Contributions

In this paper, we present a universal shape formation algorithm for self-organizing particle systems which takes as input an arbitrary shape composed of a constant number of equilateral triangles and lets the particles build that shape at a scale depending on the number of particles in the system. We limit the number of triangles to a constant because in this case every particle can store the entire shape description, which can be seen as the base case for the study of universal shape formation algorithms. More complex scenarios (i.e., a shape description is spread across multiple particles, or the shape is only implicitly defined via some algorithm) are left for future research. Our algorithm runs in $O(\sqrt{n})$ asynchronous execution rounds, where n is the number of particles in the system, provided we start from a well-initialized configuration in which the particles form a (not necessarily complete) triangle. Starting in a shape with $O(\sqrt{n})$ diameter, like a triangle, is important because if the initial shape had $\omega(\sqrt{n})$ diameter, a runtime bound of $O(\sqrt{n})$ can no longer be guaranteed, no matter which algorithm is used for the shape formation. We chose the triangle as the initial structure since it turned out to be most convenient for the design of a universal shape formation algorithm. Our algorithm relies only on local information (e.g., particles do not have ids, nor do they know n, the total number of particles, or have any sort of global coordinate system), and requires only a constant-size memory per particle.

Note that the $O(\sqrt{n})$ runtime bound is much better than the runtime bounds of $O(n)$ that were previously shown for specific shapes [10]. The reason for this is that the approach in [10] is inherently sequential while we make full use of the available parallelism in this paper. In fact, *any* shape formation algorithm would need $\Omega(\sqrt{n})$ rounds to form *any* shape that is not just a single triangle, so our runtime bound is best possible.

2. PROBLEM STATEMENT

In the *shape formation problem*, a particle system has to reconfigure into a given shape. Formally, let S be a set of faces in the standard planar embedding of G_{eqt}. Hence, S is a set of nodes in the dual graph of G_{eqt}. We say two faces of G_{eqt} are *connected* if their corresponding nodes in the dual graph of G_{eqt} are connected. We say S is connected if the induced subgraph of the dual graph of G_{eqt} is connected. We define a *shape* to be a connected set of faces that has constant size. Consider any transformation of S into a geometric shape S' defined by a translation, a rotation by a multiple of $60°$, and an isotropic scaling with the restriction that the vertices of the triangles forming S' all coincide with vertices in G_{eqt}. Consider the set of nodes $V(S') \subseteq V(G_{\text{eqt}})$ that lie on a vertex of S', on an edge of S', or inside of S'. We call $V(S')$ a *representation* of the shape S. Figure 2 illustrates this definition by an example.

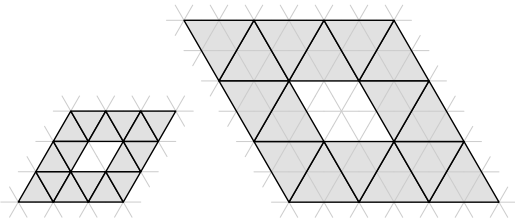

Figure 2: Two representations of a shape S consisting of 16 faces. For ease of presentation, we will often represent sets of particles via geometric shapes. In this example, each position inside a triangle, on a side of a triangle, and on a vertex of a triangle is occupied by a particle. The left part shows a representation in which the triangles coincide with the faces of G_{eqt}. Note that in this very small representation of S the hole suggested by the triangles is actually closed and therefore the representation is a solid quadrilateral. The right part shows a representation that is rotated counter-clockwise by $120°$ compared to the left representation and in which each triangle consists of four faces in G_{eqt}.

For the shape formation problem, we assume that initially all particles are contracted and they are arranged in a triangle. All particles are in the same *idle* state except for a unique *leader particle* that occupies one of the vertices of the *initial triangle* and that stores the desired shape S in its local memory (note that a leader particle could also be established by letting the particles occupying the vertices of the initial triangle engange in a competition using random coin flips that are transmitted along the sides of the triangle in a pipelined fashion). If the number of particles in the system n is not a triangular number, the last row of the initial triangle is not completely filled. For ease of presentation, we assume for now that n is a triangular number and hence the particles initially form a perfect triangle. We show how to avoid this assumption at the end of Section 5. To solve the shape formation problem, the system has to reconfigure so that the set of positions occupied by the particles is some representation of S. We allow particles in the final configuration to be expanded.

3. MOVEMENT PRIMITIVES

The shape formation algorithm reconfigures the system by using primitives for moving connected sets of particles. The simplest of these primitives moves a *chain* of particles along a path: Consider a simple directed path P of length ℓ in G_{eqt}. Let the first m nodes of P be occupied by contracted particles for some $m \leq \ell$. The goal is to let the particles *traverse* P, that is, the particles should move only through the nodes in P, they have to stay connected at all times, and in the end the particles should all be contracted and should occupy the last m nodes of P. This is achieved by letting

the particles move as a connected chain using handovers. Lemma 1 bounds the number of rounds required for the particles to traverse P. Section 3.1 provides more detail about particle chain movement.

LEMMA 1. *A chain of particles of size $m \leq \ell$ can traverse a path P of length ℓ in G_{eqt} in $O(\ell)$ rounds.*

For the remaining primitives, consider a set of contracted particles that forms a triangle in G_{eqt}. Let one of the vertices of the triangle be occupied by a distinguished particle that we call the *coordinator* of the triangle. Furthermore, let the particles on the boundary of the triangle span a path oriented counter-clockwise around the triangle.

We define four primitive operations on triangles, namely *expansion*, *contraction*, *rotation*, and *shift*. Expansion, contraction, and rotation are shown in Figure 3. A shift of a

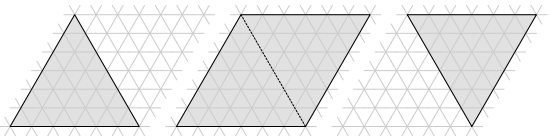

Figure 3: A triangle (left) can expand to form an expanded triangle (middle). An expanded triangle can contract back to a simple triangle (right). Concatening an expansion and a contraction effectively rotates a triangle.

triangle simply moves all particles of the triangle in a common direction by one position. A movement is initiated and controlled by the coordinator of a triangle. We give a detailed description of a local-control protocol for the expansion of a triangle in Section 3.2. Based on the general ideas presented in that section, it is easy to design protocols for the remaining operations. We have the following lemma.

LEMMA 2. *A triangle of side length ℓ can be expanded, contracted, rotated, and shifted in $O(\ell)$ rounds.*

Note that a triangle consisting of m particles has a side length of $O(\sqrt{m})$. Therefore, the primitives move such a triangle in $O(\sqrt{m})$ rounds.

3.1 Particle Chain Movement

A *movement schedule* is a sequence of particle system configurations $C_0, C_1, \ldots, C_k$. A movement schedule is called a *valid parallel schedule* (or simply *valid*) if every C_i represents a connected particle system and for every $i \geq 0$, C_{i+1} is reached from C_i in a way that for every particle p one of the following properties holds:

1. p occupies the same positions in C_i and C_{i+1}, or
2. p expands into a node that was empty in C_i, or
3. p contracts, leaving the node occupied by its tail empty in C_{i+1}, or
4. p is part of a handover with another particle p'.

Note that this means that several particles are allowed to move from C_i to C_{i+1}.

A *particle chain* is a sequence of particles $(p_1, p_2, \ldots, p_m)$ forming an arbitrary connected path of contracted or expanded particles in G_{eqt}. A *line schedule* is a valid movement schedule in which the particles initially form a particle chain and all particles move forward along the same, simple path of nodes in G_{eqt} (i.e., no node is traversed more than once by a particle) in their given order, so at the end the particles still form a particle chain of the same order as initially. Let $P = (v_1, \ldots, v_\ell)$ be the path used in a given line schedule, where v_1 is the node occupied by the tail of p_1 (the trailing particle in the particle chain) at the beginning of the schedule and v_ℓ is the node occupied by the head of p_m (the leading particle) at the end of the schedule. For any configuration C of the particle chain and any particle p_i in that chain we define its head position $h(C, i)$ to be the index of the node in P occupied by the head of p_i in C and its tail position $t(C, i)$ to be the index of the node occupied by the tail of p_i in C. Certainly, $t(C, i) \in \{h(C, i), h(C, i) - 1\}$, depending on whether p_i is contracted or expanded. For any two chain configurations C_1 and C_2 and any particle p_i we say that C_1 *dominates* C_2 w.r.t. p_i (or short, $C_1(p_i) \geq C_2(p_i)$) if and only if $h(C_1, i) \geq h(C_2, i)$ and $t(C_1, i) \geq t(C_2, i)$. Altogether, we say that C_1 *dominates* C_2 (or short, $C_1 \geq C_2$) if and only if C_1 dominates C_2 w.r.t. every particle.

We say that a given particle chain *emulates* a line schedule S if it starts in the initial configuration of S and every particle greedily aims at reaching its final position in S. That is, whenever p_i has not yet reached its final position in S, it tries to expand, contract, or use a handover whenever activated in order to advance towards its final position. However, it will only do so if this does not disconnect the particle chain.

LEMMA 3. *For any line schedule $C_0, \ldots, C_k$ that a particle chain wants to emulate and any activation sequence of the particles it holds for the configuration C'_i of the particles after round i that $C'_i \geq C_i$.*

PROOF. We will prove the lemma by induction. Initially, $C'_0 = C_0$ by assumption. Hence, suppose that for some $i \geq 0$ it holds that $C'_i \geq C_i$. Let us consider some fixed particle p_j. If $h(C'_i, j) > h(C_i, j)$ or $t(C'_i, j) > t(C_i, j)$, then certainly $C'_{i+1}(p_j) \geq C_{i+1}(p_j)$. Thus, it remains to consider the case that $h(C'_i, j) = h(C_i, j)$ and $t(C'_i, j) = t(C_i, j)$. Here, we consider the following subcases:

- If $C_{i+1}(p_j) = C_i(p_j)$, it certainly holds that $C'_{i+1}(p_j) \geq C_{i+1}(p_j)$. So for the remaining cases we assume that $C_{i+1}(p_j) > C_i(p_j)$.
- If p_j is contracted in C_i, it must be expanded in C_{i+1}, which means that p_{j+1} is expanded in C_i (or the position in front of it is empty, i.e., $j = m$). Since p_j is also contracted in C'_i, the particles must form a connected chain, and $C'_i \geq C_i$, this means that p_{j+1} must also be expanded in C'_i. No matter which of p_j and p_{j+1} is activated first, a handover will happen between them, which implies that $C'_{i+1}(p_j) \geq C_{i+1}(p_j)$. p_j cannot be involved in a handover with p_{j+1} before that because for that it would have to be expanded.
- If p_j is expanded in C_i, it must be contracted in C_{i+1}, which means that p_{j-1} is contracted in C_i (or the position behind it was empty, i.e., $j = 1$). Since p_j is also expanded in C'_i, the particles must form a connected chain, and $C'_i \geq C_i$, this means that p_{j-1} must also be contracted in C'_i. No matter which of p_j and p_{j-1} is activated first, a handover will happen between them, which implies that $C'_{i+1}(p_j) \geq C_{i+1}(p_j)$. p_j cannot be involved in a handover with p_{j-1} before that because for that it would have to be contracted.

Hence, in any case, $C'_{i+1}(p_j) \geq C_{i+1}(p_j)$ for all p_j, which proves the lemma. □

Consider a chain of contracted particles that should move along a path P of length ℓ. It is easy to see that there is a line schedule of length $O(\ell)$ that lets the particle chain traverse P. Therefore, Lemma 3 implies that by greedily moving forward the particle chain traverses P in $O(\ell)$ rounds independently of the activation order of the particles. Hence, Lemma 1 holds.

3.2 Triangle Expansion

A triangle coordinator p can *expand* its triangle to form an *expanded triangle*, i.e., two triangles of the same size of the original triangle that share a common side as illustrated in Figure 4. Let T be the set of nodes occupied by the

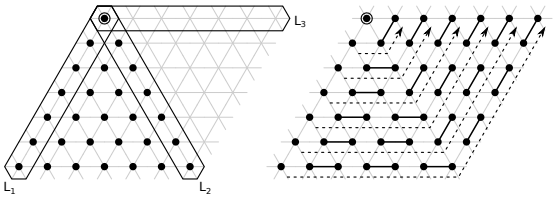

Figure 4: An example of a counter-clockwise triangle expansion. The coordinator p is highlighted by a black circle. Every row of particles expands independently as a chain (i.e., without disconnecting) around p. The resulting trajectories are shown as dashed arrows in the right part of the figure. Once every particle that did not occupy a position in L_1 is expanded, the triangle expansion is complete.

particles before the expansion and let ET be the set of positions occupied after the expansion. Let L_1 be the set of positions on the side of the initial triangle that occurs first when traversing the path on the boundary of the triangle starting from the coordinator in counter-clockwise direction. Let L_2 be the set of positions on the side of the initial triangle that corresponds to rotating L_1 counter-clockwise by 60° around p, and let L_3 be the set of positions on the side of the expanded triangle that corresponds to rotating L_1 counter-clockwise by 120°. Let a *row* of a triangle be a subset of T that forms a straight line in G_{eqt} that is parallel to the side of the initial triangle opposite of p.

An expansion is coordinated using the following token passing scheme. To initiate the expansion, p sends an *activation token* along L_1. Once a particle receives the activation token, it forwards the token along L_1 and sends a *row token* along its row. A particle that receives a row token forwards it along the row. Furthermore, it considers the particle it forwards the token to as its parent and it will follow that particle using handovers. In each row there is a unique particle that occupies L_2. Once this particle receives the expansion token it starts moving as depicted in Figure 4. In this way, all particles not on L_1 eventually expand, hence forming the expanded triangle. To determine when the expansion of the triangle is complete, the coordinator p proceeds as follows. After p sent out the activation token it sends a *validation token* along L_3. This token can only be forwarded to a specific row when the expansion of the particles in that row is complete. After the validation token completely traversed L_3, it is sent back to p along L_3. Once p receives the validation token, it knows that the expansion is completed.

Since the tokens in this scheme only traverse distances of $O(\ell)$, the corresponding parts of the algorithm can be easily realised using $O(\ell)$ rounds. Each row of the triangle acts as a chain of particles and expands independently over a distance of at most $O(\ell)$. By arguments analogous to those in the proof of Lemma 3, this movement takes $O(\ell)$ rounds. Therefore, Lemma 2 holds.

4. INTERMEDIATE STRUCTURE

The goal of this section is to reorganize the particles from the initially given triangle into an intermediate structure that simplifies the actual shape formation process. The intermediate structure we aim for is shown in Figure 5. It con-

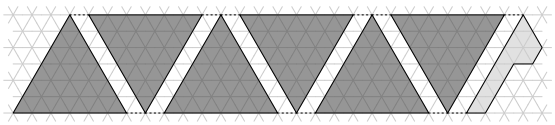

Figure 5: The intermediate structure.

sists of Δ many equilateral triangles of side length ℓ (shown in dark gray) that are arranged in a straight line and a remainder of particles (light gray) that is too small to form an additional triangle. All particles in this structure are contracted. Note that the algorithm we present in this section might actually create a structure that is slightly worse in the sense that it might leave a larger remainder of particles. We will come back to this at the end of the section; for now we assume that we can build the ideal structure.

Given a certain number of particles, the value of Δ is completely determined by the value of ℓ. Let $s = |S|$. For the intermediate structure to be useful, we want to choose ℓ such that $\frac{3}{4}s + 1 \leq \Delta \leq s - 3$. We will elaborate on these bounds in the following section; for now our focus remains on building the intermediate structure. Let L be the side length of the initial triangle. We choose $\ell = \lceil L/\lfloor c \cdot \sqrt{s} \rfloor \rceil$ with $c = 80/81$. Lemma 4 shows that this choice is appropriate for sufficiently large s and L; we omit the proof.

LEMMA 4. *If $s \geq 123$ and $L \geq 34\lfloor\sqrt{s}\rfloor$, we have that $\frac{3}{4}s + 1 \leq \Delta \leq s - 3$.*

Note that the bounds on s and L given in Lemma 4 could be lowered by a better choice of c and a tighter analysis.

We now turn to the construction of the intermediate structure. First, the particles have to determine ℓ. Note that in general no single particle can store ℓ as $\ell = \Theta(\sqrt{n})$ and the particles only have constant memory. Hence, we have to store ℓ in a distributed fashion over multiple particles. To determine ℓ we use the following token passing scheme: The leader particle, which occupies a vertex of the initial triangle, sends a *counter token* along an adjacent edge of the initial triangle. This token stores a counter that counts the number of steps it takes along the edge modulo $\lfloor c \cdot \sqrt{s} \rfloor$. Since c and s are constants, the leader can use a hard-coded table to determine $\lfloor c \cdot \sqrt{s} \rfloor$ and the token only requires constant memory. Initially, the value stored in the counter token is 0. When a particle holds the counter token while its value is 0, that particle will create a *marker token* (note that also the leader creates a marker token). Once the counter token

completely traversed the edge of the initial triangle, it is simply consumed by the particle occupying the corresponding vertex of the initial triangle. The consuming particle will then create a *terminal token*. Both the marker tokens and the terminal token travel in the opposite direction of the counter token (i.e., towards the leader). Each particle is only allowed to store either one marker token or the terminal token. The only exception to this rule is the particle that creates the terminal token. This particle might also have created a marker token. In this case, the particle can temporarily store both tokens and has to first pass the marker token and then the terminal token. Since the tokens all move towards the leader, they will eventually occupy a segment of consecutive particles starting with the leader particle and ending with the particle that holds the terminal token. To detect when this state is reached, each marker token stores a boolean value that is initially false and becomes true if either the marker token occupies the leader particle or the marker token has another marker token in front of it with its value set to true. Once the terminal token has a marker token with its value set to true in front of it, it knows that all tokens are done moving and sends a final token towards the leader to inform him that the token passing scheme has terminated. It is easy to check that the length of the segment occupied by the tokens is exactly ℓ. Furthermore, it is easy to show that the token passing scheme requires $O(\sqrt{n})$ rounds and only uses constant memory.

Once ℓ has been established, the intermediate structure can be built using a globally synchronized process. This simple process is coordinated by the leader and only uses triangle shifts and triangle rotations, as illustrated in Figure 6. For simplicity, we assume a global perspective to

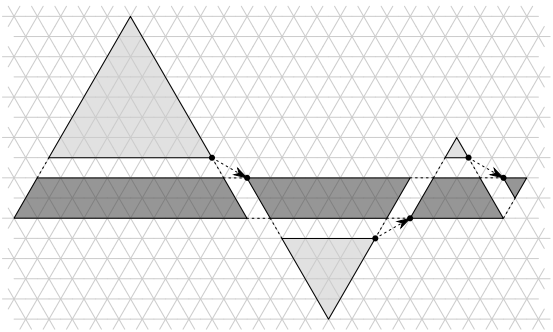

Figure 6: Construction of the intermediate structure. The parts forming the intermediate structure is shown in dark gray. The triangles that are shifted and rotated are shown in light gray. For each dotted arrow, the circles at the origin and the tip of the arrow marks the same particle before and after the respective triangle is moved. As an example, the left-most triangle is rotated clockwise by 60° around the vertex at the origin of the corresponding arrow, then it is shifted once to the right and once to the bottom-right, and finally it is rotated clockwise by 120° around the vertex at the tip of the arrow.

describe this process; a local-control protocol implementing this process is easy to design. The process is executed recursively starting with the initial triangle. It splits the triangle into a wedge of height ℓ and a smaller triangle. The wedge becomes the first part of the intermediate structure. The triangle is rotated, shifted twice, and rotated again as described in the caption of Figure 6. Then the process recurs. Note that the orientation of the rotations and the direction of the shifts changes with every recurrence. The process ends after moving the first triangle that has a side length less or equal to ℓ.

The final part of our algorithm divides the structure into triangles, each having its own coordinator, by assigning structural information to the particles. From a global perspective, the intermediate structure is simply divided into triangles that are arranged as in Figure 5. This process ends once no more triangle can be formed in this way. The process can be easily implemented locally as a traversal of the intermediate structure using token passing that is globally coordinated by the leader. Note that in this traversal the token(s) only have to move along the edges of the prospective triangles. We assume that during the process the leader counts the number of triangles Δ' that have been built, and that in the end the leader token moves back to its initial position. Concerning the number of rounds required to build the intermediate structure we have the following lemma.

LEMMA 5. *The algorithm terminates after $O(\sqrt{n})$ rounds.*

PROOF SKETCH. The recursive part of the algorithm terminates after $\lceil L/\ell \rceil = O(1)$ recursions. Each recursion requires a constant number of triangle rotations and shifts, each of which takes $O(\sqrt{n})$ rounds. The remainder of the recursive part of the algorithm as well as the final part of the algorithm that divides the intermediate structure into triangles can be implemented via token passing schemes requiring $O(\sqrt{n})$ rounds. □

As mentioned earlier, the given algorithm builds a structure that is slightly different from the ideal intermediate structure shown in Figure 5. Roughly speaking, the end of the structure is not necessarily perfectly suited for the purpose of dividing the structure into triangles of side length ℓ. Consequently, the number of triangles Δ' in the intermediate structure might be one less than the number of triangles Δ in an ideal intermediate structure, as we show in Lemma 6.

LEMMA 6. *The algorithm builds an intermediate structure containing Δ' triangles where $\Delta - 1 \leq \Delta' \leq \Delta$. The particles that are not included in any triangle are arranged as shown in Figure 7.*

PROOF. We first need the following simple observation: The end of the structure built by the algorithm can take on one of the three different forms illustrated in the first three parts of Figure 7. If the last triangle moved by the algorithm

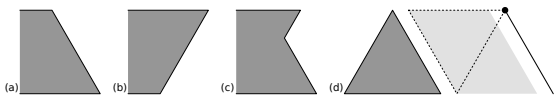

Figure 7: The end of the intermediate structure.

has side length greater or equal than $\ell - 1$, one of the forms in (a) and (b) is created. Otherwise, the end of the structure will look similar to part (c) of Figure 7.

For the rest of the proof consider part (d) of Figure 7. Without loss of generality, let the last triangle formed by the algorithm lie as shown in dark gray. The potential next

triangle, shown as a dashed outline, cannot be formed. Consequently, the vertex of that triangle marked by a circle cannot be occupied. This holds because if it were occupied all positions of the dashed triangle would be occupied, no matter which of the forms the end of the structure assumes. By the same argument, all positions on the bold line have to be unoccupied and also every position to the right of that line has to be unoccupied. Hence, all particles of the intermediate structure that are not part of a triangle have to lie in the light gray area. This area contains $\ell(\ell-1)$ positions. Since a triangle of side length ℓ contains $\ell(\ell+1)/2$ positions, the particles that are not part of any triangle could form at most one additional triangle. Therefore, the lemma holds. □

The following theorem follows immediately from the lemmas presented throughout this section.

THEOREM 1. *For $s \geq 123$ and $L \geq 34\lfloor\sqrt{s}\rfloor$, the algorithm builds an intermediate structure consisting of Δ' triangles where $\frac{3}{4}s \leq \Delta' \leq s-3$. The particles that are not included in any triangle are arranged as shown in Figure 7.*

5. SHAPE FORMATION ALGORITHM

We now turn to the actual shape formation algorithm. From this point onwards we will use the term *face* exclusively to refer to a triangular face of G_{eqt} and the term *triangle* exclusively to refer to a set of particles occupying nodes in G_{eqt} that form a triangle. The algorithm we present can be interpreted as a sequential global algorithm that is coordinated by the leader and that operates almost exclusively on triangles. Recall that for a given shape S we want to build a representation $R(S)$. We will use the shape in the left part of Figure 8 as a running example to illustrate the intermediate steps of the shape formation algorithm. Observe

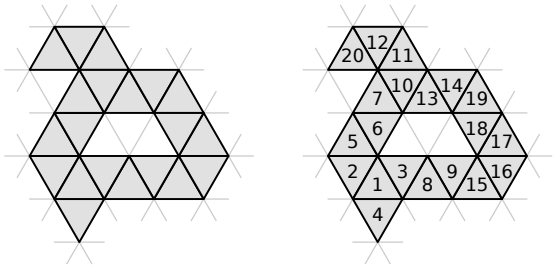

Figure 8: The running example of Section 5. The left part shows the example shape. The right part shows a corresponding construction schedule CS. The number inside a face indicates its rank in CS.

that building $R(S)$ generally requires triangles of different side lengths, namely ℓ, $\ell-1$, and $\ell-2$ (see Figure 9), since nodes in G_{eqt} can only be occupied by one particle.

Before building $R(S)$, the leader computes a *construction schedule* CS, i.e., a permutation of the faces that dictates the chronological order in which the corresponding triangles are placed. The schedule is computed backwards, i.e., the leader starts with determining which triangles to place last.

Let G_S be the subgraph of the dual graph of G_{eqt} induced by S. For the computation of CS, let D (resp. ND) be the subset of S that contains the faces that already have an assigned chronological ranking in CS (resp. have not been assigned yet). Initially, $D = \emptyset$ and $ND = S$. As long

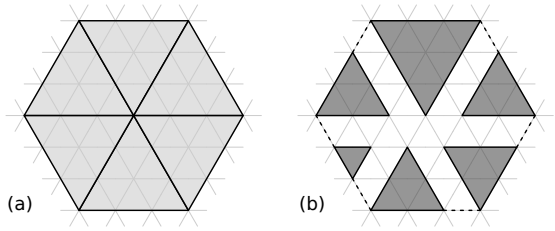

Figure 9: Part (a) shows a representation of a shape with six faces. Part (b) shows the triangles forming that representation which requires triangles of three different sizes.

as $ND \neq \emptyset$ the leader determines the next face in CS (in backwards order), removes the corresponding face from ND, and adds it to D, according to the following two rules.

1. Let G_{ND} be the subgraph of G_S induced by the nodes of ND. Mark all nodes as prospects that can be removed from G_{ND} without disconnecting the graph.

2. Instead of specifying the second rule as a graph property, it is easier and more useful to specify the condition geometrically. Interpret the nodes of G_{ND} as triangles in the Euclidean plane according to the standard embedding (see Figure 1(a)). Since G_{ND} is connected, we can combine the triangles to a single geometric shape (e.g., see Figure 10). Pick a prospect that has an edge on the outer boundary of said shape.

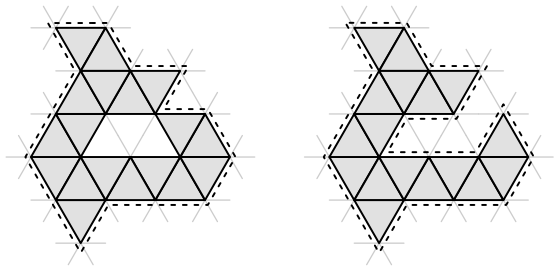

Figure 10: Two steps of the computation of CS. The left part shows the current shape after the faces of rank 20 and 19 have been chosen by the rules. The dotted line indicates the outer boundary of the current shape. Note that even though the two faces in the upper right corner are not connected, the hole of the polygon is not part of the outer boundary because the faces share a vertex. The right part shows the shape after the subsequent application of the rules (i.e. face 18 is chosen). Now the former hole is part of the boundary of the shape.

We call the chronological position of a face $a \in S$ in CS the *rank* of a in CS (i.e., the face chosen first by the rules has rank s and the one chosen last rank 1). See the right part of Figure 8 for a complete construction schedule for the running example. Informally speaking, the first rule guarantees that the schedule respects the connectivity requirements of a shape. The second rule ensures that we build $R(S)$ from the inside outwards, since outside faces are chosen early (i.e., they appear late in CS). The following lemma shows that our algorithm always computes a correct schedule.

LEMMA 7. *The algorithm assigns a rank to each face.*

PROOF. It is sufficient to show that in each step there exists at least one face of ND that fulfills both rules. We will consider the two rules in reversed order. Let ND_2 be the set of faces in ND that have an edge that lies on the outer boundary of the polygon as specified in the second rule. If there is a node in ND_2 with degree 1 in G_{ND_2} (i.e., the subgraph of G_S induced by the nodes of ND_2), that node trivially fulfills both rules. Therefore, assume that all nodes of ND_2 have a degree ≥ 2 in G_{ND_2}. Consequently, there has to exist at least one simple cycle among a subset of nodes of ND_2. In each simple cycle there has to exist at least one node with degree exactly two, since otherwise the cycle would not be on the outer boundary of the polygon. Naturally, we can remove this node from a cycle without disconnecting G_{ND}. Thus, in each step the two rules find a node that can be removed from ND. □

Moreover, we show that the C fulfills two properties that we need in order to build $R(S)$. The *prefix of length i* of a construction schedule CS is a subschedule of CS containing only the first i elements of CS.

LEMMA 8. *Consider CS as computed by our set of rules. For any prefix of length $i \in \{1, \ldots, n\}$ it holds that: (i) the subgraph of G_S induced by the prefix is connected and (ii) the face of rank i in CS has an edge that lies on the outer boundary of the shape corresponding to the prefix.*

PROOF. The first statement of the lemma follows immediately from the first rule for computing CS, and the second statement follows immediately from the second rule. □

Once CS is computed, the building of $R(S)$ does not start immediately. Section 4 pointed out that the intermediate structure is not perfect (see Figure 7), but has a remainder, i.e., particles that are not included in any triangle of the intermediate structure. Instead of ignoring the remainder, we want to incorporate it into $R(S)$ without moving it. Ultimately, we aim at building one of the two subshapes shown in Figure 11 (a) and (b) first (possibly rotated by a multiple of 60°). These subshapes have the advantage that there are two adjacent (inner) faces (shown in dark grey in Figure 11) such that both face are also adjacent to another face (light grey). The remainder will (partially) occupy the triangles

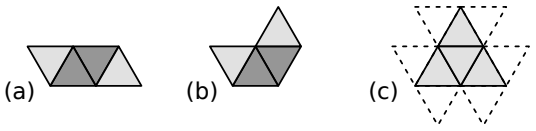

Figure 11: Three subshapes of S. We aim at building a rotated version of subshape (a) or (b) first. The subshape shown in (c) is the one we want to avoid building first. The faces with a dashed boundary are all possible locations to attach a fifth face.

corresponding to the two inner faces. We then use two triangles from the intermediate structure to represent the outer faces (see left part of Figure 12). Finally, the two triangles will use the expansion primitive to occupy all nodes in the inner triangles that are not occupied by the remainder (see

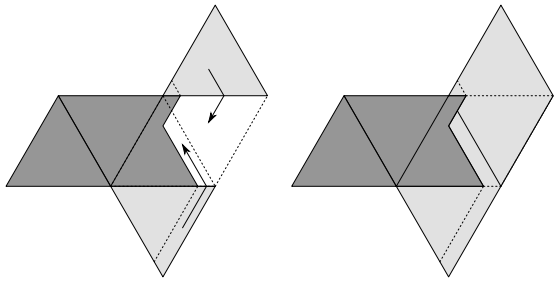

Figure 12: A conceptual sketch of how to incorporate the remainder of the intermediate structure into $R(S)$. As shown in Lemma 6 the remainder can at most occupy the nodes of two adjacent triangles of side length $\ell - 1$ (here, we abstract from concrete side lengths). The left part illustrates how the first two triangles (light grey), one of side length ℓ the other of side length $\ell - 1$, are placed adjacent to the remainder. The rows of those triangles that can expand into the white area do so (as depicted by the arrows). In the right part the rows have expanded as far as possible. The remainder and the two partially expanded triangles together form four faces of S.

right part of Figure 12). For this initial construction step two expanded triangles are necessary.

However, it is possible that the first four faces in CS do not form one of the subshapes shown in Figure 11 (a) and (b), but the one shown in Figure 11 (c) (apart from 60° rotation of those shapes there are no other possibilities, since faces have to be connected). Note that this subshape does not have two adjacent faces such that each has another adjacent face. For case (c) consider all possibilities for the fifth face in CS (i.e., faces with dashed boundary in Figure 11 (c)). No matter where it is placed, the resulting shape contains one of the two desired shapes as a subshape. Consequently, by switching the order of the first five faces in CS, we get an adjusted schedule CS' that starts with a desired subshape. In the running example, the first four faces of CS form the subshape of figure Figure 11 (c) (see right part of Figure 8); CS' is chosen accordingly (see left part of Figure 13).

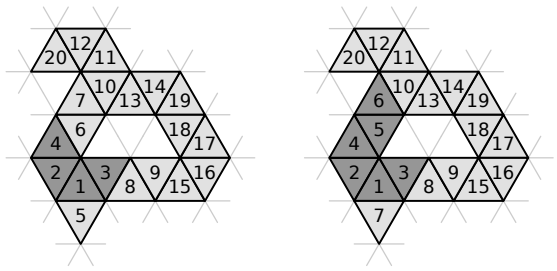

Figure 13: The two adjusted construction schedules of CS. The left part shows the adjusted construction schedule CS'. The first four faces (dark gray) now form a desired subshape. The right part illustrates CS''. The root is the subshape shown in dark gray.

Once the remainder of the intermediate structure has been incorporated as described, the algorithm performs one intermediate step before constructing the rest of $R(S)$. As already mentioned, triangles of three possible sizes are needed. Therefore, each triangle is pruned to its desired size while it is still in the intermediate structure. We will elaborate later how the leader decides which triangles to prune to a certain size; for now we focus on the pruning itself and *waste particles* arising from that pruning (i.e., particles that are removed from triangles). The pruning of a triangle is done by removing (i.e., moving away) the lower one or two rows. The number of waste particles introduced through this pruning procedure is $O(\sqrt{n})$, since the number of triangles is constant and each triangle is pruned by at most two rows. In order to incorporate the waste in $R(S)$, we want to construct an expanded triangle next (corresponding to faces ranked 5 and 6 in CS'). Note that the faces of rank 5 and 6 of CS' are not necessarily adjacent. Similar to the first four faces we can adjust CS' to get a schedule that has the desired property. To be more precise, consider the subshapes in Figure 11 (a) and (b) (i.e., the two possible subshapes we fixed for the first four triangles). If we add seven more faces to the subshape, the resulting shape will have at least another pair of adjacent faces. Therefore, it is sufficient to swap the order among the faces of rank 5 to 11 in CS' to get an adjusted schedule CS'' that has two adjacent faces at rank 5 and 6 which are connected to the faces with ranks 1 through 4, while still having the properties of Lemma 8. The schedule CS'' for the running example is shown in the right part of Figure 13. We call the substructure consisting of the first six faces in CS'' the *root* of the shape. During the construction of the root, we use particle chain movement to move the waste particles to some position that does not interfere with the construction of the root. Once the the root is constructed, we again use particle chain movement to move the waste particles to the expanded triangle forming the faces with rank 5 and 6. This completely expanded triangle then contracts as many particles as necessary to incorporate all of the waste particles. Note that the fact that we need at least three expanded triangles for the construction of the root matches the upper bound $\Delta' \leq s - 3$ from to Section 4.

Note that CS'' is oblivious to the fact that $\frac{3}{4}s \leq \Delta' \leq s - 3$. Consequently, it might be that more than the three already mentioned expanded triangles are needed to build $R(S)$. Due to our lower bound on Δ', we certainly have enough triangles. Nevertheless, the leader has to determine which pairs of faces to construct using expanded triangles. Let $\eta = s - \Delta'$, i.e., at most η many expanded triangles are needed to built $R(S)$.

LEMMA 9. *For any shape S the leader can identify at least η many disjoint pairs of adjacent faces.*

PROOF. For a shape S consider the adjusted schedule CS''. Let $r(a)$ denote the rank of a face $a \in S$ in CS''. Additionally, let the predecessor $pred(a)$ of a be the face $b \in S$ such that a and b are connected, $r(a) > r(b)$, and $r(b)$ is minimal. Note that every face has a predecessor except for the face with rank 1. Let T be the spanning tree of G_S induced by CS'', i.e., $T = (V, E^*)$ with $E^* = \{\{a, b\} \mid pred(a) = b\}$ (see left part of Figure 14).

In order to show that the leader can identify enough pairs of faces, we give the following two rules to deconstruct T and count ρ, the number of pairs found. Since we already know how to build the root, we do not apply the rules to these six nodes, i.e., the process stops once T solely consist of the six root nodes and constantly many leaves that the rules cannot be applied to.

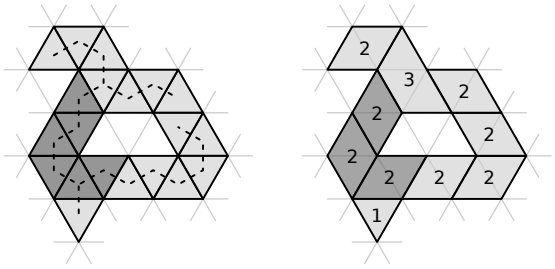

Figure 14: A graphical representation of the two intermediate steps of the proof of Lemma 9. The left part illustrates the spanning tree T (indicated by the dotted lines). The right part shors the deconstruction of T. The numbers indicate the different possibilities of the rules: (1) a single face connected to the root, (2) a pair of faces that was found by the first rule, and (3) three faces that the second rule was applied to.

1. Remove a leaf and its predecessor from T if that removal does not disconnect the remaining tree. By definition such a pair can be represented by an expanded triangle.

2. If the first rule is not applicable, each predecessor of a leaf is a predecessor for two leafs. Remove all three nodes from T, i.e., a pair and a singleton node. We can represent these three faces by one expanded and one contracted triangle.

The right part of Figure 14 shows the deconstruction of T for our running example. The lemma holds if $\eta \leq \rho$. Note that ρ is minimized if we always have to apply the second rule. Futhermore, note that at most 8 faces of S are adjacent to the root. So overall, since the root consists of 6 faces, we get that $\rho \geq \frac{1}{3}(s - 6 - 8) + 3$. Additionally, it holds that $\eta \leq \frac{1}{4}s$ due to our lower bound on Δ'. Thus, $\eta \leq \rho$ if $s \geq 20$, which holds according to the lower bound on s (see Theorem 1). □

As a consequence, the leader exactly knows how many expanded triangles it needs and has a simple rule to determine where to use an expanded triangle. The leader still has to specify which triangles have to be pruned. However, with CS'' and the information gained from the deconstruction of T, it can easily do so: It iterates over CS'' and determines for each face whether the face is represented by a triangle of size $\ell - 2$, $\ell - 1$, or ℓ. In case two adjacent faces are represented by an expanded triangle, it is important that either both faces are assigned the same size or that the lower ranked face gets the bigger size. Otherwise, the expansion would fail, since a triangle of size $\ell - 1$ has not enough particles to occupy all nodes of a triangle of size ℓ and and a triangle of size $\ell - 1$. Afterwards, the leader can prune the triangles in the intermediate structure accordingly. Thus, the building process of $R(S)$ according to CS'' is in general

straightforward. Once the root is built as described, the leader sequentially builds $R(S)$ according to CS'' by either moving a triangle with triangle rotations and triangle shifts from the intermediate structure or by expanding an already placed triangle.

Finally note that CS'' is oblivious to the position of the intermediate structure. If we keep the structure fixed in its initial position, it might get in the way of the construction of the shape or it might get walled-in, i.e., it might get completely surrounded by already placed triangles. We can avoid both of these problems as follows: Before moving a triangle in CS'', the leader can easily check whether the intermediate structure is in the way or would get walled in. If this is the case, it simply moves the entire intermediate structure, triangle by triangle, to a different position that is still adjacent to the already placed triangles but where it does not interfere with the construction. We now show that CS'' is a correct schedule, i.e., we can actually build $R(S)$ if we follow the order of the construction schedule.

LEMMA 10. *For any shape S our algorithm computes a construction schedule CS'' that allows the construction of $R(S)$.*

PROOF. First note that the intermediate structure does not interfere with the construction process since it is moved out of the way when necessary. We have already shown how to build the root. It remains to show for $i > 6$ that if the first $i-1$ faces have been built, we can always build the the face of rank i in CS''. There are two cases: Either an already built triangle has to expand or a new triangle has to be moved from the intermediate structure to the desired position. First consider the former case. There already exists a triangle a that corresponds to a face of rank strictly less than i that is adjacent to the face of rank i. Since the triangle corresponding to a has the same size or a larger size than the triangle corrsponding to the face with rank i, the i-th triangle can in fact be built using triangle expansion. Now consider the second case, i.e., a triangle has to be moved. According to property 1 of Lemma 8, the i-th face is connected to at least one of the first $i-1$ faces. According to property 2 of Lemma 8, there exists a path along which the new triangle can be moved from the intermediate structure to the desired position. Thus, we can always build $R(S)$ according to CS''. □

We conclude our section with the following theorem.

THEOREM 2. *The shape formation algorithm solves the shape formation problem in $O(\sqrt{n})$ rounds.*

PROOF. By Lemma 10 we can build $R(S)$ according to CS''. Moreover, consider the following observation.

OBSERVATION 1. *Consider two triangles of the same size that are arranged such that two vertices of one triangle are adjacent to a position of the other triangle (e.g., two triangles in the intermediate structure shown in Figure 5). We can rotate one triangle completely around the other using only a constant number of triangle rotations and shifts.*

The computation of CS with its adjustments is done locally by the leader. Building the root takes $O(\sqrt{n})$ rounds. Pruning the triangles and moving the waste particles takes $O(\sqrt{n})$ rounds according to Lemma 1. For each face in the schedule we either need to expand an already placed triangle or move a triangle to its desired position. The former case takes $O(\sqrt{n})$ rounds according to Lemma 2. For the latter case, we have to move the triangle along all other triangles in the worst case. Doing so takes at most $O(\sqrt{n})$ rounds, due to Lemma 2 and Observation 1). Additionally, it might be possible that we have to move the whole intermediate structure every time a new triangle is moved. We move the structure triangle by triangle to its new position which takes $O(\sqrt{n})$ rounds. □

The following theorem shows that our algorithm is actually *optimal* in terms of worst-case number of rounds.

THEOREM 3. *Let S be any shape except for a triangle. Our algorithm is $O(1)$-competitive against* any *algorithm that solves the shape formation problem with regards to number of rounds.*

PROOF. First note that an algorithm that solves the shape formation problem cannot control the activation order of particles, thus, every particle moves only a distance of at most 1 per round in the worst case. If S is not a triangle there exists at least one particle that has to move at distance of $\Omega(\sqrt{n})$ in order to form the desired shape. Therefore, any algorithm needs at least $\Omega(\sqrt{n})$ rounds to solve the shape formation problem in the worst case. □

Finally, we explain how to get rid of the assumption that the number of particles is a triangular number (see Section 2). As a consequence, the initial triangle of particles is not necessarily perfect anymore, i.e., the last row of the initial triangle is not completely filled with particles. Note that there are at most $O(\sqrt{n})$ particles in the last row and the leader can determine by a token passing scheme whether the last row is complete. While the intermediate structure is constructed, the incomplete row is moved to a safe location (by using the particle chain movement from Section 3) such that it does not interfere with the process described in Section 4. During the construction of the representation of S we move the incomplete row together with the intermediate structure if necessary. We incorporate the row into $R(S)$ early on by moving it into the third expanded triangle we construct, which is similar to the way the waste particles are handled. There is enough space for the waste particles and the incomplete row since both amounts are in $O(\sqrt{n})$ and the expanded triangle can incorporate $\Omega(n)$ many particles. During the entire algorithm, the incomplete row only has to move a distance of $O(\sqrt{n})$, which is in line with the given runtime bound.

6. CONCLUSION

We presented a universal shape formation algorithm for self-organizing particle systems which takes as input an arbitrary shape composed of a constant number of equilateral triangles and lets the particles build that shape at a scale depending on the number of particles in the system. For future work, it would be interesting to investigate more complex descriptions of the shape that has to be built: a shape description of non-constant size could be spread across multiple particles, or the shape could be defined only implicitly via some algorithm. Furthermore, initial configurations of diameter $O(\sqrt{n})$ other than the triangle could be considered.

7. ACKNOWLEDGMENTS

This work was partially supported by the NSF under the Awards CCF-1353089 and CCF-1422603. Furthermore, it was partially supported by DFG grant SCHE 1592/3-1.

8. REFERENCES

[1] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[2] D. Arbuckle and A. Requicha. Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations. *Autonomous Robots*, 28(2):197–211, 2010.

[3] Z. J. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *International Journal of Robotics Research*, 23(9):919–937, 2004.

[4] A. Chavoya and Y. Duthen. Using a genetic algorithm to evolve cellular automata for 2d/3d computational development. In *8th annual conference on genetic and evolutionary computation*, pages 231–232, 2006.

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

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

[7] S. Das, P. Flocchini, N. Santoro, and M. Yamashita. On the computational power of oblivious robots: forming a series of geometric patterns. In *Proceedings of PODC 2010*, pages 267–276, 2010.

[8] X. Defago and S. Souissi. Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity. *Theoretical Computer Science*, 396(1-3):97–112, 2008.

[9] E. D. Demaine, M. J. Patitz, R. T. Schweller, and S. M. Summers. Self-assembly of arbitrary shapes using RNAse enzymes: Meeting the kolmogorov bound with small scale factor (extended abstract). In *Proceedings of STACS '11*, pages 201–212, 2011.

[10] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proceedings of NANOCOM 2015*, 2015.

[11] Z. Derakhshandeh, R. Gmyr, T. Strothmann, R. A. Bazzi, A. W. Richa, and C. Scheideler. Leader election and shape formation with self-organizing programmable matter. In *DNA Computing and Molecular Programming (DNA 21)*, pages 117–132, 2015.

[12] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1):412–447, 2008.

[13] K. Imai, Y. Kasai, Y. Sonoyama, C. Iwamoto, and K. Morita. Self-reproduction and shape formation in two and three dimensional cellular automata with conservative constraints. In *Proceedings of the Eighth International Symposium on Artificial Life and Robotics*, pages 377–380, 2003.

[14] O. Michail. Terminating distributed construction of shapes and patterns in a fair solution of automata. In *Proceedings of PODC 2015*, pages 37–46, 2015.

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

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

[17] P. W. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

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

[19] M. Rubenstein and W.-M. Shen. Automatic scalable size selection for the shape of a distributed robotic collective. In *Proceedings of IROS 2010*, pages 508–513. IEEE, 2010.

[20] N. Schiefer and E. Winfree. Universal computation and optimal construction in the chemical reaction network-controlled tile assembly model. In *DNA Computing and Molecular Programming (DNA 21)*, pages 34–54, 2015.

[21] J. L. Schiff. *Cellular automata: a discrete view of the world*, volume 45. John Wiley & Sons, 2011.

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

[23] D. Woods. Intrinsic universality and the computational power of self-assembly. In *Proceedings of MCU 2013*, pages 16–22, 2013.

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

[25] K. Yeom. Bio-inspired automatic shape formation for swarms of self-reconfigurable modular robots. In *Proceedings of BIC-TA 2010*, pages 469–476, 2010.

[26] K. Yeom and B.-J. You. Agent based morphological approach for collaborative shape formation of self-organizable unmanned aerial vehicles. In *Proceedings of HPCC_EUC 2013*, pages 85–92, 2013.

[27] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robotics Automation Magazine*, 14(1):43–52, 2007.