On the Shape Containment Problem Within the Amoebot Model with Reconfigurable Circuits

Matthias Artmann
Paderborn University, Germany

Andreas Padalkin
Paderborn University, Germany

Christian Scheideler
Paderborn University, Germany

Abstract

In *programmable matter*, we consider a large number of tiny, primitive computational entities called *particles* that run distributed algorithms to control global properties of the particle structure. *Shape formation* problems, where the particles have to reorganize themselves into a desired shape using basic movement abilities, are particularly interesting. In the related *shape containment* problem, the particles are given the description of a shape S and have to find maximally scaled representations of S within the initial configuration, without movements. For example, if S is a triangle, they have to identify the largest subsets of particles that already form a triangle. While the shape formation problem is being studied extensively, no attention has been given to the shape containment problem, which may have additional uses besides shape formation, such as detecting structural flaws.

In this paper, we consider the shape containment problem within the *geometric amoebot model* for programmable matter, using its *reconfigurable circuit extension* to enable the instantaneous transmission of primitive signals on connected subsets of particles. We first prove a lower runtime bound of $\Omega(\sqrt{n})$ synchronous rounds for the general problem, where n is the number of particles. Then, we present simple and efficient primitives for identifying subsets that form the desired shape. Using these primitives, we construct a large class of shapes which we call *snowflakes*. This class contains, among others, all shapes composed of parallelograms and hexagons, and the class of *star convex* shapes. Let k be the maximum scale of the considered shape in a given amoebot structure. If the shape is star convex, we solve it within $\mathcal{O}(\log^2 k)$ rounds. If it is a snowflake but not star convex, we solve it within $\mathcal{O}(\sqrt{n}\log n)$ rounds.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Self-organization; Theory of computation → Computational geometry

Keywords and phrases Programmable matter, amoebot model, reconfigurable circuits, shape containment

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.7

Related Version *Full Version*: `https://arxiv.org/abs/2501.16892` [4]

Funding This work was supported by the DFG Project SCHE 1592/10-1.

Acknowledgements We thank Daniel Warner for his guidance and helpful discussions.

1 Introduction

Programmable matter envisions a material that can change its physical properties in a programmable fashion [25] and react to external stimuli. It is typically viewed as a system of many identical micro-scale computational entities called *particles*. Potential application areas include minimally invasive surgery, maintenance, exploration, and manufacturing. While significant progress is being made in the field of micro-scale robotics [5, 26], the fundamental capabilities and limitations of such systems are studied in theory using various models [24].

In the *amoebot model* of programmable matter, the particles are called *amoebots* and are placed on the nodes of a graph. We assume that the occupied nodes form a connected subgraph. Since information can only travel through the edges of the graph, there is a natural lower bound of $\Omega(D)$ for many problems, where D is the diameter of this subgraph.

Motivated by this, we consider the *reconfigurable circuit extension* of the model, which allows better results with reasonable modifications. In this extension, the amoebots can construct simple communication networks called *circuits* on connected subgraphs of the structure and broadcast primitive signals on these circuits instantaneously. This allows polylogarithmic solutions to many problems, e.g., leader election, consensus, shape recognition, and shortest path forest construction [12, 20, 21].

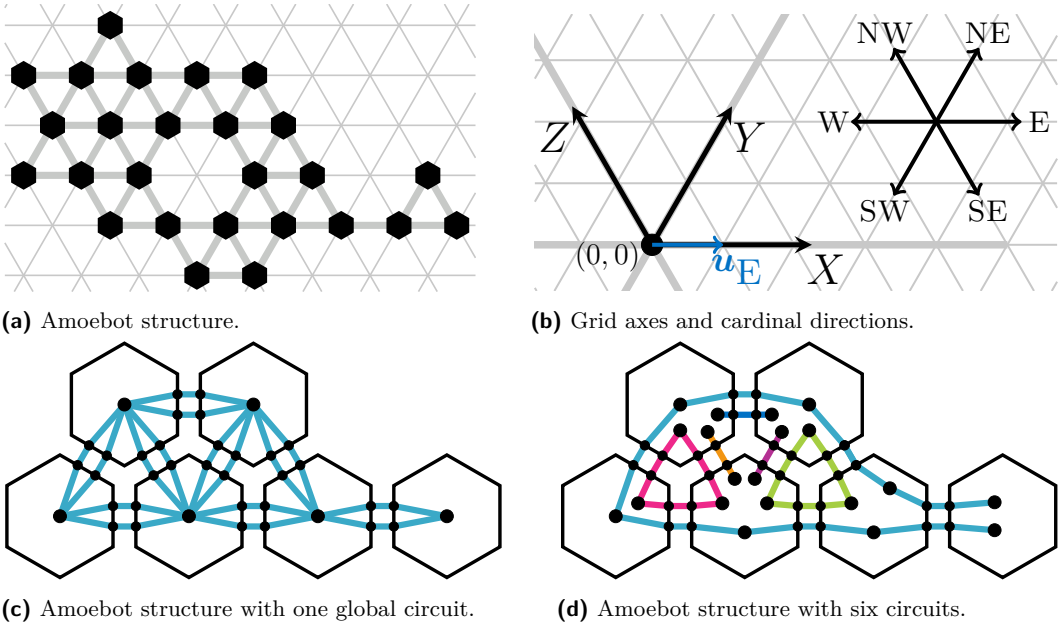
The *shape formation problem*, where the amoebot structure has to reconfigure itself into a given shape, is a standard problem of particular interest [24]. In this paper, we study the related *shape containment problem*: Given a shape S, the amoebots must find the maximum scale at which S can be placed within their structure and identify all valid placements at this scale. This can be useful for shape formation by *self-disassembly*, i.e., disconnecting all amoebots that are not part of a selected placement of the shape from the structure [13, 14]. The problem can also be interpreted as a discrete variant of the *polygon containment problem* in classical computational geometry, which has been studied extensively [6, 23]. To our knowledge, there are no distributed solutions that apply to the amoebot model.

1.1 Geometric Amoebot Model

We use the *geometric amoebot model* for programmable matter, as proposed in [9]. Using the terminology from the recent *canonical* model description [8], we assume *common direction* and *chirality*, *constant-size memory*, and a fully synchronous scheduler, making it *strongly fair*. We describe the model in sufficient detail here and refer to [8, 9] for more information.

The geometric amoebot model places n particles called *amoebots* on the infinite regular triangular grid graph $G_\Delta = (V_\Delta, E_\Delta)$ (see Fig. 1a). This is the typical graph used in the amoebot literature [9, 12, 19–21]. Each amoebot occupies one node, and each node is occupied by at most one amoebot. We identify each amoebot with the grid node it occupies to simplify the notation. Thereby, we define the *amoebot structure* $A \subset V_\Delta$ as the set of occupied nodes. We assume that A is finite and its induced subgraph of G_Δ is connected.

Each amoebot has a local *compass* identifying one of its incident grid edges as the East direction, and a *chirality* defining its local sense of clockwise rotation. We assume that all amoebots share both. This is not a restrictive assumption because a common compass and chirality can be established efficiently using circuits [12]. The amoebots are controlled in a distributed fashion by identical and anonymous finite state machines. In particular, the amount of memory per amoebot is *constant*, i.e., fixed by the algorithm/state machine running on each amoebot and independent of the number of amoebots in the structure. This means that, for example, unique identifiers for all amoebots cannot be stored. A computation proceeds in *fully synchronous rounds*. In each round, all amoebots act and change their states simultaneously based on their current state and their received signals (see Section 1.2). We measure the time complexity of an algorithm by the number of rounds it requires until all amoebots reach a terminal state. Although the model allows amoebots to perform movements, we only consider static amoebot structures in this paper.



(a) Amoebot structure.

(b) Grid axes and cardinal directions.

(c) Amoebot structure with one global circuit.

(d) Amoebot structure with six circuits.

Figure 1 (a) shows an amoebot structure in the triangular grid. Amoebots are represented by black hexagons and neighboring amoebots are connected by thick edges. (b) shows the axes and cardinal directions in the triangular grid and the unit vector in the East direction. (c, d) illustrate the reconfigurable circuit extension for $c_\phi = 2$. Amoebots are drawn as hexagons, pins are black circles on their borders, and partition sets are drawn as black circles inside the hexagons. The partition sets are connected to the pins they contain. Partition sets in the same circuit have lines of the same color.

1.2 Reconfigurable Circuit Extension

The *reconfigurable circuit extension* [12] places c_ϕ *external links* on every edge connecting two adjacent amoebots $u, v \in A$. The endpoints of an external link are called *pins*. For each link, one pin is owned by u and one is owned by v. The constant c_ϕ is an algorithm design parameter and is the same for all amoebots. In this paper, we use $c_\phi = 2$, which is the least number of pins required by the PASC algorithm [12], a central primitive for our results.

Let $P(u)$ be the set of pins owned by amoebot $u \in A$. Each amoebot u computes a partitioning of $P(u)$ into a set $\mathcal{Q}(u)$ of non-empty, pairwise disjoint subsets $Q \subseteq P(u)$ such that $\bigcup_{Q \in \mathcal{Q}(u)} Q = P(u)$. The subsets $Q \in \mathcal{Q}(u)$ are called *partition sets* and $\mathcal{Q}(u)$ is called the *pin configuration* of u. The amoebot's algorithm defines how the pin configuration is computed in every round. Let $\mathcal{Q} := \bigcup_{u \in A} \mathcal{Q}(u)$ be the set of all partition sets in the amoebot structure. Two partition sets $Q \in \mathcal{Q}(u)$ and $Q' \in \mathcal{Q}(v)$ of neighboring amoebots u and v are *connected* if there is an external link with one pin in Q and one pin in Q'. Let $E_\mathcal{Q}$ be the set of these connections. We call each connected component C of the undirected graph $G_\mathcal{Q} := (\mathcal{Q}, E_\mathcal{Q})$ a *circuit* (see Figs. 1c, d). An amoebot u is part of a circuit C if C contains at least one partition set of u. Note that multiple partition sets of an amoebot u may be contained in the same circuit without u being aware of this due to its lack of global information.

During its activation, each amoebot can establish an arbitrary new pin configuration and send primitive signals called *beeps* on any selection of its partition sets. A beep is broadcast to the circuit containing the partition set it was sent on. All partition sets in that circuit

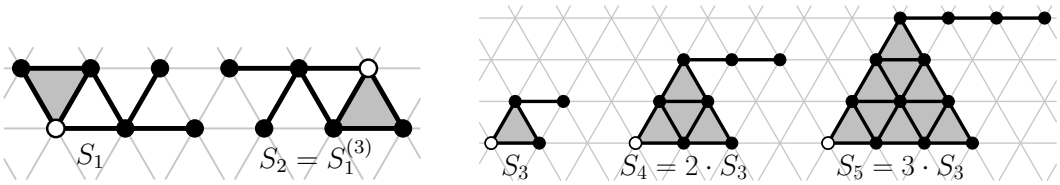


Figure 2 Examples of shapes and shape transformations. Each shape is identified by the grid nodes, edges, and faces it contains. The origin of each shape is highlighted in white (we translate some shapes for better visibility). S_1 and S_2 are equivalent, and each contains one face and one hole. The shapes S_3, S_4 and S_5 illustrate the scaling operation.

receive the beep at the beginning of the next round. An amoebot can tell which of its partition sets have received a beep, but it has no information on the identity, location, or number of beeping amoebots. We model all communication between amoebots with circuits.

1.3 Problem Statement and Our Contribution

Consider the embedding of G_Δ into $\mathbb{R}^2$ such that the grid's faces form unit triangles and one grid node is placed at the plane's *origin* $(0,0) \in \mathbb{R}^2$ (see Fig. 1b). We obtain three coordinate axes, X, Y, Z, and six *cardinal directions* $\mathcal{D} = \{\mathrm{E}, \mathrm{NE}, \mathrm{NW}, \mathrm{W}, \mathrm{SW}, \mathrm{SE}\}$. Let $\boldsymbol{u}_d$ denote the unit vector in direction $d \in \mathcal{D}$.

A *shape* $S \subset \mathbb{R}^2$ is a finite union of nodes, edges, and faces of the embedded grid graph (see Fig. 2, c.f. [10,12]). Edges contain their endpoints and faces contain their enclosing edges. A shape must be connected, but it may contain *holes*, i.e., $\mathbb{R}^2 \setminus S$ might not be connected. This shape definition matches the one used in [10] for shape formation and extends the definition used in [12] for shape recognition.

We define *translation*, *rotation*, and *scaling* operations on a shape S as follows. For $t \in \mathbb{R}^2$, we denote S translated by t by $S + t := \{p + t \mid p \in S\}$. This is a valid shape if and only if t is the position of a grid node, i.e., a linear combination of unit vectors in the cardinal directions with integer coefficients. We can therefore write $t \in V_\Delta$, identifying the position with the unique grid node occupying it. For $r \in \mathbb{Z}$, let $S^{(r)}$ denote S after r counter-clockwise rotations by 60° around the origin of $\mathbb{R}^2$. Note that $r \in \{0, \dots, 5\}$ is sufficient to represent all distinct rotations of a shape in the grid. Let $k \in \mathbb{R}$ be a *scale factor*, then we define $k \cdot S := \{k \cdot p \mid p \in S\}$ to be the shape S scaled by k. We only consider positive integer scale factors to ensure that the resulting set is a valid shape. If S is *minimal*, i.e., there is no scale factor $0 < k' < 1$ such that $k' \cdot S$ is a valid shape, then the integer scale factors cover all possible scales of S that produce valid shapes (see Lemma 1 in [10]). Two shapes are *equivalent* if one can be obtained from the other by a rigid motion, i.e., a composition of a translation and a rotation.

Let $V(S) \subset V_\Delta$ denote the set of grid nodes contained in S. For convenience, we assume $(0,0) \in V(S)$ for any given shape S and call this node the *origin* of S. This property is invariant under rotation and scaling as defined above. A *valid placement* of a shape S in an amoebot structure A is an amoebot $p \in A$ with $V(S + p) \subseteq A$ (identifying p with the grid node it occupies). Let $\mathcal{V}(S, A) \subseteq A$ denote the set of all valid placements of S in A. The *maximum scale* of S in A is the largest scale $k \in \mathbb{N}_0$ such that there is a valid placement of $k \cdot S^{(r)}$ in A for some $r \in \mathbb{Z}$:

$$k_{\max}(S, A) := \sup \left\{ k \in \mathbb{N}_0 \;\middle|\; \exists\, r \in \mathbb{Z} : \mathcal{V}\left(k \cdot S^{(r)}, A\right) \neq \emptyset \right\}$$

$k_{\max}$ is well-defined because $0 \cdot S$ is at most a single node for every shape S, which fits into

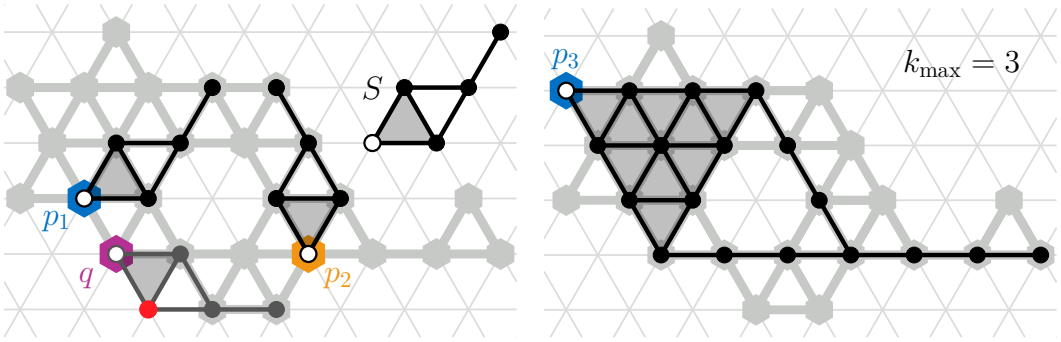


Figure 3 Valid placements of a shape S in the amoebot structure A from Fig. 1a. p_1 and p_2 are valid placements of S and $S^{(1)}$, respectively. q is not a valid placement of $S^{(5)}$ because one node of $V(S^{(5)} + q)$ is not contained in A. p_3 is the only valid placement of $3 \cdot S^{(5)}$ and there are no valid placements for any scale $k > 3$, so we have $k_{\max}(S, A) = 3$. A shape containment algorithm allows p_3 to determine $p_3 \in \mathcal{V}(k_{\max} \cdot S^{(5)}, A)$ and rules out all other amoebots and rotations.

any non-empty amoebot structure A. We obtain $k_{\max}(S, A) = \infty$ if and only if S does not contain any edges. If S and A are clear from the context, we will write $k_{\max} = k_{\max}(S, A)$. See Fig. 3 for illustration.

We define the *shape containment problem* as follows: Let S be a shape (containing the origin). An algorithm solves the shape containment problem instance (S, A) for an amoebot structure A if it terminates eventually and at the end,

1. all amoebots know whether $k_{\max} \in \{0, \infty\}$ (meaning that all or none of the amoebots are valid placements), and
2. if $k_{\max} \in \mathbb{N}$, then for every $r \in \{0, \dots, 5\}$, each amoebot knows whether it is contained in $\mathcal{V}(k_{\max} \cdot S^{(r)}, A)$.

The algorithm solves the shape containment problem for S if it solves the shape containment instances (S, A) for all finite connected amoebot structures A. Note that the shape S is fixed for an algorithm, i.e., it is encoded in the state machine in some way, and its size is constant. The algorithm may use an equivalent shape S' instead of S itself.

There are two key challenges in solving the shape containment problem. First, the amoebots have to find the maximum scale $k_{\max}$. We approach this problem by testing individual scale factors for valid placements until $k_{\max}$ is fixed. We call this part of an algorithm the *scale factor search*. Second, for a given scale k and a rotation r, the valid placements of $k \cdot S^{(r)}$ have to be identified. This means that every amoebot $p \in A$ must collect information on all amoebots in $V(k \cdot S^{(r)} + p)$. p then has to mark itself as a valid placement if and only if all of these amoebots exist. In some cases, we initially view all amoebots as *placement candidates* and then *eliminate* candidates that can be ruled out as valid placements. We call this part the *valid placement search* procedure. Since the information about missing amoebots naturally comes from the boundaries of the structure, the main difficulty here is distributing this information efficiently to all affected amoebots and not ruling out any valid placements.

In this paper, we present sublinear time solutions for the shape containment problem using circuits. As a motivation, we first prove a lower bound for the general case that holds even if the maximum scale is known, by creating a communication bottleneck for an example shape. Next, we present scale factor search approaches using distributed memory, applying a binary search where it is possible. We then introduce simple and efficient primitives for placement search algorithms based on combining and transforming valid placements of lines, thereby

constructing increasingly complex shapes. Using earlier results for reconfigurable circuits, these primitives run in at most logarithmic time. By combining our primitives, we obtain the class of *snowflake* shapes, which contains a variety of complex and practically relevant shapes. For example, it contains all convex shapes, all shapes that are composed of parallelograms of the same orientation, all shapes composed of hexagons, and combinations (e.g., unions) of these and other shapes. The definition of this class allows more types of shapes to be combined and integrated when a valid placement search procedure is given for them. Our shape containment solution for snowflakes takes a sublinear number of rounds. Further, we show that for the set of *star convex* shapes, which is contained in the snowflake class, a binary search for the scale factor even leads to a polylogarithmic solution. Surprisingly, the binary search approach is *only* directly applicable to star convex shapes. We only give high-level explanations of our primitives and defer the technical details and proofs to the full version of the paper [4]. Selected proofs can also be found in the appendix.

1.4 Related Work

The authors of [12] demonstrated the potential of their reconfigurable circuit extension with algorithms solving the leader election, compass alignment, and chirality agreement problems within $\mathcal{O}(\log n)$ rounds, w.h.p.[1] They also presented efficient solutions for some exact shape recognition problems: Given common chirality, an amoebot structure can determine whether it matches a scaled version of a given shape composed of edge-connected faces in $\mathcal{O}(1)$ rounds. Without common chirality, convex shapes can be detected in $\mathcal{O}(1)$ rounds and parallelograms with linear or polynomial side ratios can be detected in $\Theta(\log n)$ rounds, w.h.p.

The PASC algorithm was introduced in [12] and refined in [21], and it allows amoebots to compute distances along chains. It has become a central primitive in the reconfigurable circuit extension, as it was used to construct spanning trees, detect symmetry, and identify centers and axes of symmetry in polylogarithmic time, w.h.p. [21]. The authors in [20] used it to solve the single- and multi-source shortest path problems, requiring $\mathcal{O}(\log \ell)$ rounds for a single source and ℓ destinations and $\mathcal{O}(\log n \log^2 k)$ rounds for k sources and any number of destinations. The PASC algorithm also plays a crucial role in this paper (see Sec. 2.2.2).

The authors in [11] studied the capabilities of a generalized circuit communication model that directly extends the reconfigurable circuit model to general graphs. They provided polylogarithmic time algorithms for various common graph construction (minimum spanning tree, spanner) and verification problems (minimum spanning tree, cut, Hamiltonian cycle etc.). Additionally, they presented a generic framework for translating a type of lower bound proof from the widely used CONGEST model into the circuit model, demonstrating that some problems are hard in both models, while others can be solved much faster with circuits. For example, checking whether a graph contains a 5-cycle takes $\Omega(n/\log n)$ rounds in general graphs, even with circuits, while the verification of a connected spanning subgraph can be done with circuits in $\mathcal{O}(\log n)$ rounds w.h.p., which is below the lower bound shown in [22].

In the context of computational geometry, the basic polygon containment problem was studied in [6], focusing on the case where only translation and rotation are allowed. The problem of finding the largest copy of a convex polygon inside some other polygon was discussed in [23] and [1], for example. An example for the problem of placing multiple polygons inside another without any polygons intersecting each other is given by [17]. More

[1] An event holds *with high probability* (w.h.p.) if it holds with probability at least $1 - n^{-\delta}$, where the constant δ can be made arbitrarily large.

recently, the authors in [16] showed lower bounds for several polygon placement cases under the kSUM conjecture. For example, assuming the 5SUM conjecture, there is no $\mathcal{O}((p+q)^{3-\varepsilon})$-time algorithm for any $\varepsilon > 0$ that finds a largest copy of a simple polygon P with p vertices that fits into a simple polygon Q with q vertices under translation and rotation. Perhaps more closely related to our setting (albeit centralized) is an algorithm that solves the problem of finding the largest area parallelogram inside of an object in the triangular grid, where the object is a set of edge-connected faces [2].

2 Preliminaries

This section introduces elementary algorithms for the circuit extension from previous work.

2.1 Coordination and Synchronization

As mentioned before, we assume that all amoebots share a common compass direction and chirality. This is a reasonable assumption because the authors of [12] have presented randomized algorithms establishing both in $\mathcal{O}(\log n)$ rounds, w.h.p.

We often want to synchronize amoebots, for example, when different parts of the structure run independent instances of an algorithm simultaneously. For this, we can use a *global circuit*: Each amoebot connects all of its pins into a single partition set. The resulting circuit spans the whole structure and allows the amoebots which are not yet finished with their procedure to inform all other amoebots by sending a beep. When no beep is sent, all amoebots know that all instances of the procedure are finished. Due to the fully synchronous scheduler, we can establish the global circuit periodically at predetermined intervals.

2.2 Chains and Chain Primitives

A *chain* of amoebots with length $m-1$ is a sequence of m amoebots $C = (p_0, \dots, p_{m-1})$ where all subsequent pairs p_i, p_{i+1}, $0 \leq i < m-1$, are neighbors, each amoebot except p_0 knows its predecessor, and each amoebot except p_{m-1} knows its successor. In this paper, we only consider *simple* chains without multiple occurrences of the same amoebot. By letting each amoebot decide whether it connects a pin toward its predecessor with a pin toward its successor, we can easily establish circuits along such chains.

2.2.1 Binary Operations

The constant memory limitation of amoebots makes it difficult to deal with non-constant data, particularly numbers that can increase with n. However, we can use amoebot chains to implement a distributed memory by letting each amoebot on the chain store one bit of a binary number, as demonstrated in [7, 21]. Using circuits, we can implement efficient comparisons and arithmetic operations between two operands stored on the same chain.

▶ **Lemma 1.** *Let $C = (p_0, \dots, p_{m-1})$ be an amoebot chain such that each amoebot p_i stores two bits a_i and b_i of the integers a and b, where $a = \sum_{i=0}^{m-1} a_i 2^i$ and $b = \sum_{i=0}^{m-1} b_i 2^i$. Within $\mathcal{O}(1)$ rounds, the amoebots on C can compare a to b and compute the first m bits of $a+b$, $a-b$ (if $a \geq b$), $2 \cdot a$, and $\lfloor a/2 \rfloor$ and store them on the chain. Within $\mathcal{O}(m)$ rounds, the amoebots on C can compute the first m bits of $a \cdot b$, $\lfloor a/b \rfloor$, and $a \bmod b$ and store them on the chain.*

Lemma 1 is a minor improvement over the algorithms presented in [21]. Additionally, individual amoebots can execute simple binary operations online on *streams* of bits:

▶ **Lemma 2.** *Let p be an amoebot that receives two numbers a, b as* bit streams*, i.e., it receives the bits a_i and b_i in the i-th iteration of some procedure, for $i = 0, \dots, m$. Then, p can compute bit c_i of $c = a + b$ or $c = a - b$ (if $a \geq b$) in the i-th iteration and determine the comparison result between a and b by iteration m, with only constant overhead per iteration.*

2.2.2 The PASC Algorithm

A particularly useful algorithm in the reconfigurable circuit extension is the *Primary-And-Secondary-Circuit* (PASC) algorithm, first introduced in [12]. We omit the details of the algorithm and only outline its relevant properties. Please refer to [21] for details.

▶ **Lemma 3** ([12, 21]). *Let $C = (p_0, \dots, p_{m-1})$ be a chain of m amoebots. The PASC algorithm, executed on C with start point p_0, performs $\lceil \log m \rceil$ iterations within $\mathcal{O}(\log m)$ rounds. In iteration $j = 0, \dots, \lceil \log m \rceil - 1$, each amoebot p_i computes the j-th bit of its distance i to the start of the chain, i.e., p_i computes i as a bit stream.*

The PASC algorithm is especially useful with binary counters. It allows us to compute the length of a chain, which is received by the last amoebot in the chain and can be stored in binary on the chain itself (letting the last amoebot send the received bits as beeps on a circuit spanning the chain). Furthermore, given some binary counter storing a distance ℓ and some amoebot chain $C = (p_0, \dots, p_{m-1})$, each amoebot p_i can compare i to ℓ by receiving the bits of ℓ on a global circuit in sync with the iterations of the PASC algorithm on C.

▶ **Lemma 4.** *Let $C = (p_0, \dots, p_{m-1})$ be a chain in an amoebot structure A and let a value $\ell \in \mathbb{N}_0$ be stored in some binary counter of A. Within $\mathcal{O}(\log \min\{\ell, m\})$ rounds, every amoebot p_i can compare i to ℓ. The procedure can run simultaneously on any set of edge-disjoint chains with length $\leq m - 1$.*

3 A Lower Bound for Finding Valid Placements

We first show a lower bound that demonstrates a central difficulty arising in the shape containment problem. For a simple example shape S_Ω (see Fig. 4), we show that even if the maximum scale is known, identifying all valid placements of S_Ω can require $\Omega(\sqrt{n})$ rounds due to communication bottlenecks.

▶ **Theorem 5.** *There exists a shape S_Ω such that for any choice of origin and every amoebot algorithm $\mathcal{A}$ that terminates after $o(\sqrt{n})$ rounds, there exists an amoebot structure A for which the algorithm does not compute $\mathcal{V}(k_{\max}(S_\Omega, A) \cdot S_\Omega, A)$, even if $k_{\max}$ is known.*

The proof idea is as follows (see Appendix A for details): Consider an amoebot structure A consisting of two parts, X and Y, which are only connected by a single edge. In one round, one of at most 2^{c_ϕ} different signals can be sent via the c_ϕ external links on that edge. Suppose that for $k = k_{\max}(S_\Omega, A)$, all valid placements of $k \cdot S_\Omega$ lie in X, and each placement's required node set reaches into Y. By adding and removing amoebots from Y, we can control which amoebots in X are valid placements. The idea for our lower bound proof is to construct 2^k distinct patterns of valid placements by only altering Y. To identify the valid placements correctly, the amoebots in X must distinguish between all of these patterns using information that must cross the edge between X and Y. Since c_ϕ is a constant, this takes $\Omega(k)$ rounds. We show such a construction for any $k \geq 2$, with $n = \mathcal{O}(k^2)$, and for any choice of origin in S_Ω. In the remainder of this paper, we explore for which shapes these arbitrary patterns of valid placements do not occur, and present efficient procedures exploiting this property.

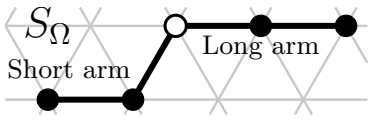


Figure 4 An example shape for which the valid placement search is bounded below by $\Omega(\sqrt{n})$.

This lower bound holds for the problem of finding valid placements. Since every algorithm solving the shape containment problem must compute all valid placements for the maximum scale, the bound also holds for the general shape containment problem. Further, note that the lower bound of $\Omega(D)$ in the amoebot model without circuits, where $D = \Omega(\sqrt{n})$ is the diameter of A, is caused by the *distance* information can travel per time step. In contrast, this lower bound is caused by the limited *bandwidth* of the circuits in some amoebot structures.

4 Scale Factor Search

As outlined before, our shape containment algorithms consist of two search procedures. The first is a *scale factor search* that determines which scales have to be checked to find the maximum scale. The second procedure is a *valid placement search* that identifies all valid placements of $k \cdot S^{(r)}$ for all $r \in \{0, \dots, 5\}$, given the scale k in a binary counter.

Consider some shape S and an amoebot structure A with a binary counter storing an upper bound $K \geq k_{\max}(S, A)$. The simple *linear search* procedure runs valid placement checks for the scales $K, K-1, \dots, 1$ and accepts when the first valid placement is found. If no placement is found in any iteration, we have $k_{\max} = 0$.

▶ **Lemma 6.** *Let S be a shape and A an amoebot structure with a binary counter storing an upper bound $K \geq k_{\max}(S, A)$. Given a valid placement search procedure for S, the amoebots compute $k_{\max}(S, A)$ in at most K iterations, running the placement search for scales $K, K-1, \dots, k_{\max}$ and with constant overhead per iteration.*

When using the linear search method, finding a small upper bound K is essential for reducing the runtime. One way of obtaining K for S is to use a simpler shape S' with $S' \subseteq S$ for which $k_{\max}$ can be computed efficiently. Since every valid placement of S is also a valid placement of S', we have $k_{\max}(S, A) \leq k_{\max}(S', A) =: K$. Moving on, some shapes permit a faster search method based on an inclusion relation between different scales.

▶ **Definition 7.** *We call a shape S* self-contained *if for all scales $k < k'$, there exist a translation $t \in \mathbb{R}^2$ and a rotation $r \in \{0, \dots, 5\}$ such that $k \cdot S^{(r)} + t \subseteq k' \cdot S$.*

For self-contained shapes, finding no valid placements at scale k immediately implies $k_{\max}(S, A) < k$, which allows us to apply a *binary search*: Starting with scale $k = 1$, we double the scale factor until no valid placements are found at some scale $K = 2^j$, which we then use as the upper bound for a binary search between 1 and K.

▶ **Lemma 8.** *Let S be a self-contained shape and let A be an amoebot structure with a binary counter large enough to store $k_{\max} = k_{\max}(S, A)$. Given a valid placement search procedure for S, the amoebots can compute $k_{\max}$ within $\mathcal{O}(\log k_{\max})$ iterations such that each iteration runs the valid placement search once for some scale $k \leq 2 \cdot k_{\max}$ and has constant overhead.*

5 Efficient Placement Search Procedures

In this section, we introduce placement search procedures allowing amoebots to identify valid placements of shapes when their scale is given in a binary counter. To cover all rotations, we simply repeat the placement search six times with rotated directions. We only provide high-level descriptions of the algorithms. Please refer to the full paper [4] for details. Our procedures heavily rely on combining the PASC algorithm with binary operations on bit streams (see Lemma 4). A simple and natural way to establish the required chains is using *segments*:

▶ **Definition 9.** *Let $W \in \{X, Y, Z\}$ be a grid axis. A* (W)-segment *is a connected set of grid nodes on a line parallel to W. Let $C \subseteq V_\Delta$, then a* maximal W-segment of C *is a finite W-segment $M \subseteq C$ that cannot be extended with nodes from C on either end. The* length *of a finite segment M is $|M| - 1$.*

For example, chains on maximal segments of the amoebot structure A can be constructed easily once a direction has been agreed upon: All amoebots on a segment identify their chain predecessor and successor by checking the existence of neighbors on the direction's axis. The start and end points of the segment are the unique amoebots lacking a neighbor in one or both directions. Using chains on segments, we can find valid placements of *line shapes*.

▶ **Definition 10.** *A* line shape $\mathrm{L}(d, \ell)$ *is a shape consisting of $\ell \in \mathbb{N}_0$ consecutive edges extending in direction d from the origin. For $\ell = 0$, the shape only contains the origin point.*

Lines are the fundamental primitive shapes we will use to construct much more complex shapes. Our placement search procedure for lines runs the PASC algorithm to compute indices along maximal amoebot segments and compares them to the given line length ℓ, which is transmitted on a global circuit. Exactly the amoebots whose chain index, i.e., distance to the end of the segment, is at least ℓ are valid placements of the line. If the line shape L has length ℓ and the current scale is k, we compute $\ell' = k \cdot \ell$ in the counter storing k and run the procedure for length ℓ'. Note that for line shapes, the boundary amoebots holding the information which placements are invalid lie on the same segment as the affected placement candidates, enabling this simple solution.

▶ **Lemma 11.** *Let $L = \mathrm{L}(d, \ell)$ be a line shape and let A be an amoebot structure that stores ℓ in some binary counter and where all amoebots agree on d. Within $\mathcal{O}(\log \min\{\ell, n\})$ rounds, the amoebots can compute $\mathcal{V}(L, A)$.*

Recall that for any two shapes S, S' with $S \subseteq S'$, $k_{\max}(S, A)$ is an upper bound for $k_{\max}(S', A)$. Thus, the maximum scale of an edge $\mathrm{L}(d, 1)$ is a natural upper bound on the maximum scale of any non-trivial shape. In particular, any longest segment in the amoebot structure provides sufficient memory to store the scale values that must be considered. To use this fact, we establish binary counters on *all* amoebot segments (on all axes) and use them simultaneously, deactivating counters if they run out of memory.

We now discuss efficient operations on valid placements that allow us to determine the valid placements of a transformed shape. The first, simple operation is the *union* of shapes. Given the valid placements $C_1 = \mathcal{V}(S_1, A)$ and $C_2 = \mathcal{V}(S_2, A)$ of two shapes S_1 and S_2, the amoebots in A can find the valid placements of $S' = S_1 \cup S_2$ in a single round: Due to the relation $\mathcal{V}(S', A) = C_1 \cap C_2$, each amoebot only has to check whether it is a valid placement of both shapes. Observe that S' is always connected because we assume that S_1 and S_2 contain the origin, and the location of the origin in S_1 and S_2 directly affects the shape S'.

5.1 The Minkowski Sum Primitive

Next, we consider the *Minkowski sum* of a shape with a line.

▶ **Definition 12.** *Let S_1, S_2 be two shapes, then their* Minkowski sum *is defined as*

$$S_1 \oplus S_2 := \{p_1 + p_2 \mid p_1 \in S_1, p_2 \in S_2\}.$$

The resulting subset of $\mathbb{R}^2$ is a valid shape, and if both shapes contain the origin, then their sum also contains the origin. Observe that for any shape S and any line $\mathrm{L}(d, \ell)$, we have

$$V(S \oplus \mathrm{L}(d, \ell)) = \bigcup_{i=0}^{\ell} V(S + i \cdot \boldsymbol{u}_d),$$

i.e., $V(S \oplus \mathrm{L}(d, \ell))$ consists of $\ell + 1$ consecutive copies of $V(S)$. This means that an amoebot p is a valid placement of $S' = S \oplus \mathrm{L}(d, \ell)$ if and only if it is a valid placement of the line $L = \mathrm{L}(d, \ell)$ and for every amoebot $q \in V(L + p)$, q is a valid placement of S. Suppose the amoebots already know the set C of valid placements of S and store ℓ in a binary counter. Then, they can find the valid placements of $S \oplus L$ by running the placement search for L within C, treating the amoebots in $A \setminus C$ as if they did not exist, except for synchronization.

▶ **Lemma 13.** *Let S be an arbitrary shape, $L = \mathrm{L}(d, \ell)$ a line, and A an amoebot structure storing ℓ and a scale k in some binary counter. Given d and the set $C = \mathcal{V}(k \cdot S, A)$, the amoebots can compute $\mathcal{V}(k \cdot (S \oplus L), A)$ within $\mathcal{O}(\log \min\{k \cdot \ell, n\})$ rounds.*

Observe that this already yields efficient valid placement search procedures for shapes like parallelograms $\mathrm{L}(d_1, \ell_1) \oplus \mathrm{L}(d_2, \ell_2)$ and unions thereof.

5.2 The Translation Primitive

Consider some shape S, its valid placements $C = \mathcal{V}(S, A)$, a direction $d \in \mathcal{D}$, and a distance $\ell \in \mathbb{N}$ such that $\ell \cdot \boldsymbol{u}_d \in S$. Now, let $S' = S - \ell \cdot \boldsymbol{u}_d$, then S' only differs from S in the location of its origin, and the valid placements of S' are just translated versions of the placements of S, i.e., $\mathcal{V}(S', A) = C + \ell \cdot \boldsymbol{u}_d$. Given the valid placements of S, we can compute the placements of S' with a procedure that translates the placement information from C. Our *translation primitive* can do this efficiently for shapes with the following property:

▶ **Definition 14.** *Let S be a shape and $W \in \{X, Y, Z\}$ a grid axis. The* minimal axis width of S on W *(or W-*width*) is the infimum of the length of any maximal line segment in S that is parallel to W. We call S (W-)*wide *or* wide on W *if its W-width is at least* 1.

For example, the minimal axis width of a triangular face is 0 for all axes due to its corners, and the W-width of $\mathrm{L}(d, \ell)$ is ℓ when d is parallel to W and 0 otherwise. Further, observe that for any shape S, direction d on axis W, and $\ell \in \mathbb{N}$, the W-width of $S \oplus \mathrm{L}(d, \ell)$ is at least ℓ.

Now, consider a W-wide shape S and let $C = \mathcal{V}(k \cdot S, A)$ for some scale $k \in \mathbb{N}$. Note that $k \cdot S$ has a W-width of at least k. Let $q \in A \setminus C$ be an *invalid* placement of $k \cdot S$, then q must be part of a W-segment of $A \setminus C$ that has length at least k or is bounded by unoccupied nodes on at least one side. This is because any unoccupied node $x \in V_\Delta \setminus A$ that makes q invalid lies in $V(k \cdot S + q)$. Since $V(k \cdot S)$ consists of W-segments of length at least k, x causes a similar segment which contains q to be invalid, as any placement in this segment is also invalidated by x.

For translating the valid placements C of $k \cdot S$ by k steps in a direction d parallel to W, suppose that $\mathrm{L}(d,1) \subseteq S$ (otherwise, we add this edge as part of the shape transformation). Then, we can reduce the problem to translating a single segment of length at least k. For this, we run a procedure in every maximal W-segment M of A simultaneously. Suppose we want to translate some segment C_M in M. It suffices to translate the start p_1 and endpoint p_2 of C_M, identifying $p_1' = p_1 + k \cdot \boldsymbol{u}_d$ and $p_2' = p_2 + k \cdot \boldsymbol{u}_d$. To identify p_i', the amoebots in M run the PASC algorithm with p_i as the start point while transmitting k on a global circuit. Each amoebot compares its distance to p_i to k, and the amoebot with distance equal to k becomes p_i'. Afterward, the amoebots in M establish a chain circuit between p_1' and p_2', and send a beep that is received by all amoebots on the translated segment. Due to the size of the considered segments C_M, this procedure can run for almost all segments within M simultaneously without interference. The translation primitive moves the origin of a W-wide shape on the axis W and adds the line connecting the old and new origin locations.

▶ **Lemma 15.** *Let S be a W-wide shape and A an amoebot structure that stores a scale k in a binary counter and knows $\mathcal{V}(k \cdot S, A)$. Given a direction d on axis W and $\ell \in \mathbb{N}$, the amoebots can compute $\mathcal{V}(k \cdot ((S + \ell \cdot \boldsymbol{u}_d) \cup \mathrm{L}(d,\ell)), A)$ within $\mathcal{O}(\ell \cdot \log \min\{k \cdot \ell, n\})$ rounds.*

5.3 The Triangle Primitive

Finally, we show how to find the valid placements of *triangles* using the above primitives. Line shapes are scaled edges and triangles are scaled faces.

▶ **Definition 16.** *Let $\mathrm{T}(d,1)$ be the shape consisting of the single triangular face spanned by the edges $\mathrm{L}(d,1)$ and $\mathrm{L}(d',1) = \mathrm{L}(d,1)^{(1)}$. We define general* triangle shapes *as $\mathrm{T}(d,\ell) := \ell \cdot \mathrm{T}(d,1)$ for $\ell \in \mathbb{N}_{>1}$ and call ℓ the* side length *or* size *of $\mathrm{T}(d,\ell)$.*

To construct the valid placements of $T = \mathrm{T}(d,\ell)$, we cover $V(T)$ with the union of three parallelograms. The side lengths of the parallelograms are $\ell' = \lfloor \ell/2 \rfloor$ and $\ell'' = \lfloor \ell/2 \rfloor + (\ell \bmod 2)$. Two parallelograms are translated to reach the corners of the triangle. The algorithm can be summarized as follows: The amoebots first compute ℓ' and ℓ'' on the binary counter storing ℓ. Then, they run the line primitive to compute the valid placements of three lines of length ℓ' or ℓ''. Next, they apply the Minkowski sum primitive to the three sets of line placements, resulting in the valid placements of three parallelograms. To translate two of the parallelograms, they run the translation primitive, which is applicable since the parallelograms have a width of at least ℓ' on the translation axis (for a translation by $\ell'' = \ell' + 1$, the valid placements can be moved by one more step in a single round using only local communication). Finally, they intersect the valid placements of all three parallelograms (union operation) to obtain the valid placements of the triangle.

▶ **Lemma 17.** *Let $T = \mathrm{T}(d,\ell)$ be a triangle shape and let A be an amoebot structure that stores ℓ in some binary counter. The amoebots can compute $\mathcal{V}(T, A)$ within $\mathcal{O}(\log \min\{\ell, n\})$ rounds.*

6 Shape Classification

We now generalize the idea of the triangle primitive to define the class of *snowflake* shapes, whose valid placements can be found efficiently using our primitives. Then, we discuss interesting subsets of these shapes and combine our valid placement search procedures with suitable scale factor search approaches to solve the shape containment problem.

6.1 Snowflake Shapes

Combining the primitives discussed in the previous section, we obtain the following class of shapes. Our definition identifies shapes with trees such that every node in the tree represents a shape and every edge represents a composition or transformation of shapes.

▶ **Definition 18.** *A* snowflake tree *is a finite, non-empty tree $T = (V_T, E_T)$ with three node labeling functions $\tau : V_T \to \{\mathrm{L}, \mathrm{T}, \cup, \oplus, +\}$, $d : V_T \to \mathcal{D}$ and $\ell : V_T \to \mathbb{N}_0$ that satisfies the following constraints. Every node $v \in V_T$ represents a shape S_v such that:*

- *If $\tau(v) = \mathrm{L}$, then v is a leaf node and $S_v = \mathrm{L}(d(v), \ell(v))$* (line node).
- *If $\tau(v) = \mathrm{T}$, then v is a leaf node and $S_v = \mathrm{T}(d(v), \ell(v))$, where $\ell(v) > 0$* (triangle node).
- *If $\tau(v) = \cup$, then $S_v = \bigcup_{i=1}^{m} S_{u_i}$, where $u_1, \ldots, u_m$ are the children of v and $m \geq 2$* (union node).
- *If $\tau(v) = \oplus$, then $S_v = S_u \oplus \mathrm{L}(d(v), \ell(v))$, where u is the unique child of v and $\ell(v) > 0$* (sum node).
- *If $\tau(v) = +$, then $S_v = (S_u + \ell(v) \cdot \boldsymbol{u}_{d(v)}) \cup \mathrm{L}(d(v), \ell(v))$, where u is the unique child of v, S_u has a minimal axis width > 0 on the axis of $d(v)$, and $\ell(v) > 0$* (translation node).

Let $r \in V_T$ be the root of T, then we say that S_r is the snowflake shape *represented by T.*

Note that this definition constrains the location of a snowflake's origin. Because algorithms for the shape containment problem can place the origin of a shape freely, we may include equivalent shapes in the class of snowflakes. Our algorithms always place the origin by the definition. Fig. 5 shows examples for snowflakes and non-snowflake shapes.

To compute the valid placements of a snowflake, we apply the primitives from Section 5 following a *topological ordering* of the shape's tree representation. This way, the amoebots first compute valid placements of primitive shapes (lines and triangles), and then they apply the union, Minkowski sum, and translation operations successively until they arrive at the valid placements of the shape represented by the root node. This sequence of operations is encoded in the state machine of each amoebot; we say that the amoebots "have access" to the shape and its tree representation.

Consider a snowflake S represented by a tree $T = (V_T, E_T)$ with labelings $\tau(\cdot)$, $d(\cdot)$ and $\ell(\cdot)$. We assume that each amoebot has access to a representation of T and a topological ordering of T from the leaves to the root. The amoebots compute $\mathcal{V}(k \cdot S, A)$ as follows:

1. For every leaf $v \in V_T$, perform the placement search for the scaled primitive $\mathrm{L}(d(v), k \cdot \ell(v))$ or $\mathrm{T}(d(v), k \cdot \ell(v))$ represented by v. Let $C(v) \subseteq A$ be the resulting set of valid placements.
2. Process each non-leaf node $v \in V_T$ in the topological ordering as follows:
   a. If $\tau(v) = \cup$, then set $C(v) = \bigcap_{i=1}^{m} C(u_i)$, where the u_i are the child nodes of v.
   b. If $\tau(v) = \oplus$, let u be the unique child of v. Run the procedure from Section 5.1 to compute $C(v)$ from $C(u)$.
   c. If $\tau(v) = +$, let u be the unique child of v. Run the procedure from Section 5.2 to compute $C(v)$ from $C(u)$.
3. Let $r \in V_T$ be the root of T. Terminate with success ($k_{\max} \in \mathbb{N}$) and report the valid placements as $\mathcal{V}(k \cdot S, A) = C(r)$ if $C(r) \neq \emptyset$, otherwise terminate with failure ($k_{\max} = 0$).

▶ **Lemma 19.** *Let A be an amoebot structure storing a scale k in some binary counter and let S be a snowflake. Given an encoding of the tree T of S with a topological ordering, the amoebots can compute $\mathcal{V}(k \cdot S^{(r)}, A)$ for $r \in \{0, \ldots, 5\}$ within $\mathcal{O}(\log \min\{k, n\})$ rounds.*

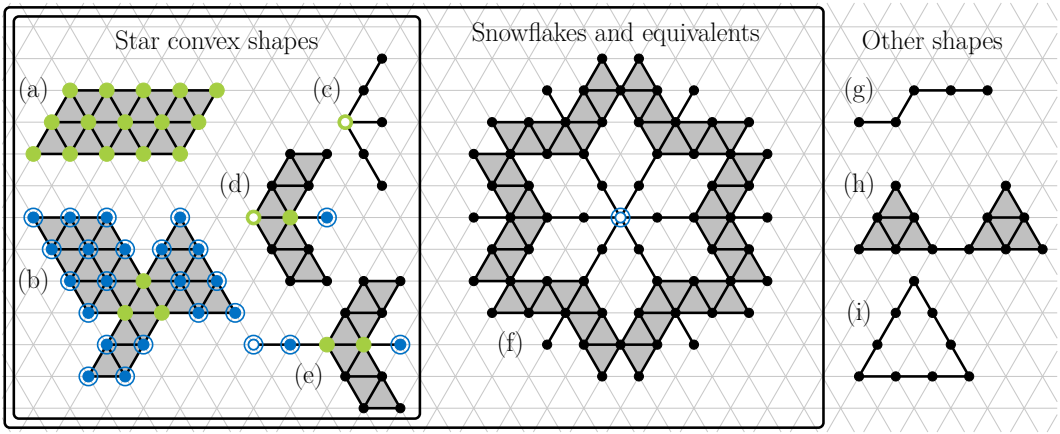


Figure 5 Examples of snowflakes, star convex shapes, and non-snowflake shapes. Green nodes indicate star convex shape centers, encircled blue nodes are possible snowflake origins, and origins used for operations have a white center. All center nodes are also snowflake origins. Shape (a) is convex and shape (b) demonstrates that not all snowflake origins must be center nodes. Shape (c) is the union of three lines, (d) is the Minkowski sum of (c) and $\mathrm{L}(\mathrm{E}, 1)$, shape (e) is the union of $\mathrm{L}(\mathrm{E}, 2)$ and (d) $+ 2 \cdot \boldsymbol{u}_{\mathrm{E}}$, and shape (f) is the union of six rotated variants of shape (e). (g) is the example shape S_Ω for the lower bound from Section 3.

The class of snowflake shapes is the result of our primitive operations and is therefore somewhat artificial. However, it contains a large variety of shapes, and its modular definition provides a framework for constructing efficient valid placement search algorithms for more shapes. For example, any shape that is composed of parallelograms with the same orientation is a snowflake, even if the parallelograms are only connected at the corners. This is because a parallelogram $\mathrm{L}(d_1, \ell_1) \oplus \mathrm{L}(d_2, \ell_2)$ is both W_1-wide and W_2-wide, where W_i is the axis of d_i. Therefore, the translation primitive can be used to move the shape's origin to any position in the parallelogram, after which the union primitive allows the addition of another parallelogram at this position. Since this maintains the width properties, it can be repeated until all parallelograms have been added to the shape. A similar method is possible for shapes composed of hexagons because hexagons are even wide on all three axes. Constructing shapes out of building blocks like parallelograms is a popular approach for shape formation in grid-based modular robot systems [3, 19]. Shapes in square grid models can be represented by parallelograms in the triangular grid. Furthermore, any valid placement search algorithm for non-snowflake shapes can be integrated into this framework as a primitive to expand the set of shapes.

6.2 Star Convex Shapes

To combine our valid placement search procedure with a scale factor search, we would like to know for which shapes the binary search approach is applicable. Recall from Section 4 that at least *self-contained* shapes permit a binary search. In this section, we characterize this class of shapes exactly as the *star convex* shapes.

▶ **Definition 20.** *A shape S is* star convex *if it is hole-free and contains a* center node c *such that for every $v \in V(S)$, all shortest paths from c to v in G_Δ are contained in S.*

For example, all convex shapes are star convex since all their nodes are centers. It is easy to see that star convex shapes are self-contained: When placing the origin of S on a center node, no translation is necessary and $k \cdot S \subseteq k' \cdot S$ holds for all $k, k' \in \mathbb{N}, k < k'$. We can

even show that *only* star convex shapes are self-contained. As the authors of [15] point out in their extensive survey on *starshaped sets* (see p. 1007), the results in [18] show that these two properties are equivalent in much more general settings when rotations are omitted.

▶ **Theorem 21.** *A shape is self-contained if and only if it is star convex.*

The main proof and supporting lemmas can be found in Appendix B. In our context, this equivalence implies that the efficient binary search can only be applied directly to star convex shapes. For any non-star convex shape S, there exist an amoebot structure A and scale factors k, k', $k < k'$, such that $\mathcal{V}(k \cdot S^{(r)}, A) = \emptyset$ for all r but $\mathcal{V}(k' \cdot S^{(r)}, A) \neq \emptyset$ for some r; consider, e.g., $A = V(k' \cdot S)$ for sufficiently large k and k'. The proof of Theorem 21 shows that this always holds for infinitely many scales k, k'.

6.3 Shape Containment Solutions

Finally, we obtain algorithms solving the shape containment problem by combining valid placement search procedures with suitable scale factor search methods. First, we observe that star convex shapes are snowflakes:

▶ **Lemma 22.** *Every star convex shape S is equivalent to a snowflake. If its origin is a center node, S itself is a snowflake.*

This implies that we can solve the shape containment problem for a star convex shape S in just $\mathcal{O}(\log^2 k_{\max}(S, A))$ rounds by using a binary search. For a snowflake S that is not star convex, we apply the linear search approach, which runs $\mathcal{O}(K)$ valid placement searches when given some upper bound $K \geq k_{\max}(S, A)$. To obtain this upper bound, we use the observation that a snowflake without faces will always be a union of line shapes meeting at the origin, and therefore star convex. Thus, a non-star convex snowflake must contain at least one face, so $K = k_{\max}(\mathrm{T}(d, 1), A)$ is a suitable upper bound (for any $d \in \mathcal{D}$). Since $\mathrm{T}(d, 1)$ is star convex, K can be computed in $\mathcal{O}(\log^2 K)$ rounds. And since a triangle fills an area, we have $|V(k \cdot \mathrm{T}(d, 1))| = \Theta(k^2)$, so we have an upper bound of $K = \mathcal{O}(\sqrt{n})$. Together, we obtain the following theorem (see Appendix C for the proof):

▶ **Theorem 23.** *Let A be an amoebot structure and S a snowflake shape. Given a tree representation of S, the amoebots can compute $k = k_{\max}(S, A)$ in a binary counter and determine $\mathcal{V}(k \cdot S^{(r)}, A)$ for all $r \in \{0, \dots, 5\}$ within $\mathcal{O}(\log^2 k)$ rounds if S is star convex and $\mathcal{O}(K \log K)$ rounds otherwise, where $k \leq K = k_{\max}(\mathrm{T}(E, 1), A) = \mathcal{O}(\sqrt{n})$.*

7 Conclusion and Future Work

In this paper, we introduced the shape containment problem for the amoebot model of programmable matter and presented sublinear solutions using reconfigurable circuits. We showed that for some shapes, there is a lower bound of $\Omega(\sqrt{n})$ rounds due to the arbitrary distribution of valid and invalid placements, even if the maximum scale is known. Using efficient methods of transferring information using circuits, we constructed the class of *snowflake* shapes that can be solved in sublinear time and contains a large variety of shapes. For the subset of shapes that are *star convex*, we showed how to solve the problem in polylogarithmic time and proved that binary search is not generally applicable to other shapes.

It is unclear how non-snowflake shapes can be characterized and what exactly distinguishes shapes that are affected by the lower bound from shapes with more efficient solutions, such as star convex shapes. It would be interesting to explore whether the lower bound can be improved when including the scale factor search. Naturally, efficient solutions for arbitrary shapes or other shape classes are of interest as well.

The related problems of finding the smallest scale at which a given shape contains the amoebot structure, as well as finding scales and placements with maximal overlap and minimal difference, could also be investigated to support shape formation algorithms. To expand the set of possible applications further, one could examine other, non-uniform scaling behaviors, perhaps allowing the shapes to maintain fine details at larger scales, similar to fractal shapes.

References

1 Pankaj K. Agarwal, Nina Amenta, and Micha Sharir. Largest Placement of One Convex Polygon Inside Another. *Discrete & Computational Geometry*, 19(1):95–104, 1998. doi:10.1007/PL00009337.

2 Md Abdul Aziz Al Aman, Raina Paul, Apurba Sarkar, and Arindam Biswas. Largest Area Parallelogram Inside a Digital Object in a Triangular Grid. In Reneta P. Barneva, Valentin E. Brimkov, and Giorgio Nordo, editors, *Combinatorial Image Analysis - 21st International Workshop (IWCIA)*, volume 13348 of *LNCS*, pages 122–135, Cham, 2022. Springer. doi:10.1007/978-3-031-23612-9_8.

3 Greg Aloupis, Sébastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristán, and Stefanie Wuhrer. Reconfiguration of Cube-Style Modular Robots Using O(log n) Parallel Moves. In Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, editors, *Algorithms and Computation*, volume 5369 of *Lecture Notes in Computer Science*, pages 342–353, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-92182-0_32.

4 Matthias Artmann, Andreas Padalkin, and Christian Scheideler. On the shape containment problem within the amoebot model with reconfigurable circuits, 2025. doi:10.48550/arXiv.2501.16892.

5 Ahmed Amine Chafik, Jaafar Gaber, Souad Tayane, Mohamed Ennaji, Julien Bourgeois, and Tarek El Ghazawi. From Conventional to Programmable Matter Systems: A Review of Design, Materials, and Technologies. *ACM Comput. Surv.*, 56(8):210:1–210:26, 2024. doi:10.1145/3653671.

6 Bernard Chazelle. The Polygon Containment Problem. *Advances in Computing Research*, 1(1):1–33, 1983.

7 Joshua J. Daymude, Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Christian Scheideler, and Andréa W. Richa. Convex Hull Formation for Programmable Matter. In Nandini Mukherjee and Sriram V. Pemmaraju, editors, *21st International Conference on Distributed Computing and Networking*, ICDCN '20, pages 1–10, New York, NY, USA, 2020. ACM. doi:10.1145/3369740.3372916.

8 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The canonical amoebot model: Algorithms and concurrency control. *Distributed Computing*, 36(2):159–192, 2023. doi:10.1007/s00446-023-00443-3.

9 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Brief Announcement: Amoebot - A New Model for Programmable Matter. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 220–222, Prague, Czech Republic, 2014. ACM. doi:10.1145/2612669.2612712.

10 Giuseppe A. Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. Shape formation by programmable particles. *Distributed Computing*, 33(1):69–101, 2020. doi:10.1007/s00446-019-00350-6.

11 Yuval Emek, Yuval Gil, and Noga Harlev. On the Power of Graphical Reconfigurable Circuits. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2024.22.

12 Michael Feldmann, Andreas Padalkin, Christian Scheideler, and Shlomi Dolev. Coordinating Amoebots via Reconfigurable Circuits. *Journal of Computational Biology*, 29(4):317–343, 2022. doi:10.1089/cmb.2021.0363.

13 Melvin Gauci, Radhika Nagpal, and Michael Rubenstein. Programmable Self-disassembly for Shape Formation in Large-Scale Robot Collectives. In Roderich Groß, Andreas Kolling, Spring Berman, Emilio Frazzoli, Alcherio Martinoli, Fumitoshi Matsuno, and Melvin Gauci, editors, *Distributed Autonomous Robotic Systems: The 13th International Symposium*, volume 6 of *Springer Proceedings in Advanced Robotics*, pages 573–586, Cham, 2018. Springer. doi:10.1007/978-3-319-73008-0_40.

14 Kyle W. Gilpin. *Shape Formation by Self-Disassembly in Programmable Matter Systems*. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2012.

15 G. Hansen, I. Herburt, H. Martini, and M. Moszyńska. Starshaped sets. *Aequationes mathematicae*, 94(6):1001–1092, 2020. doi:10.1007/s00010-020-00720-7.

16 Marvin Künnemann and André Nusser. Polygon Placement Revisited: (Degree of Freedom + 1)-SUM Hardness and an Improvement via Offline Dynamic Rectangle Union. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 3181–3201, Alexandria, VA, USA, 2022. SIAM. doi:10.1137/1.9781611977073.124.

17 Thiago de Castro Martins and Marcos de Sales Guerra Tsuzuki. Simulated annealing applied to the irregular rotational placement of shapes over containers with fixed dimensions. *Expert Systems with Applications*, 37(3):1955–1972, 2010. doi:10.1016/j.eswa.2009.06.081.

18 P. McMullen. Sets homothetic to intersections of their translates. *Mathematika*, 25(2):264–269, 1978. doi:10.1112/S0025579300009505.

19 Andreas Padalkin, Manish Kumar, and Christian Scheideler. Reconfiguration and Locomotion with Joint Movements in the Amoebot Model. In Arnaud Casteigts and Fabian Kuhn, editors, *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, volume 292 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SAND.2024.18.

20 Andreas Padalkin and Christian Scheideler. Polylogarithmic Time Algorithms for Shortest Path Forests in Programmable Matter. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *43rd ACM Symposium on Principles of Distributed Computing*, PODC '24, pages 65–75, New York, NY, USA, 2024. ACM. doi:10.1145/3662158.3662776.

21 Andreas Padalkin, Christian Scheideler, and Daniel Warner. The Structural Power of Reconfigurable Circuits in the Amoebot Model. In Thomas E. Ouldridge and Shelley F. J. Wickham, editors, *28th International Conference on DNA Computing and Molecular Programming (DNA 28)*, volume 238 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DNA.28.8.

22 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed Verification and Hardness of Distributed Approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012. doi:10.1137/11085178X.

23 Micha Sharir and Sivan Toledo. Extremal polygon containment problems. *Computational Geometry*, 4(2):99–118, 1994. doi:10.1016/0925-7721(94)90011-6.

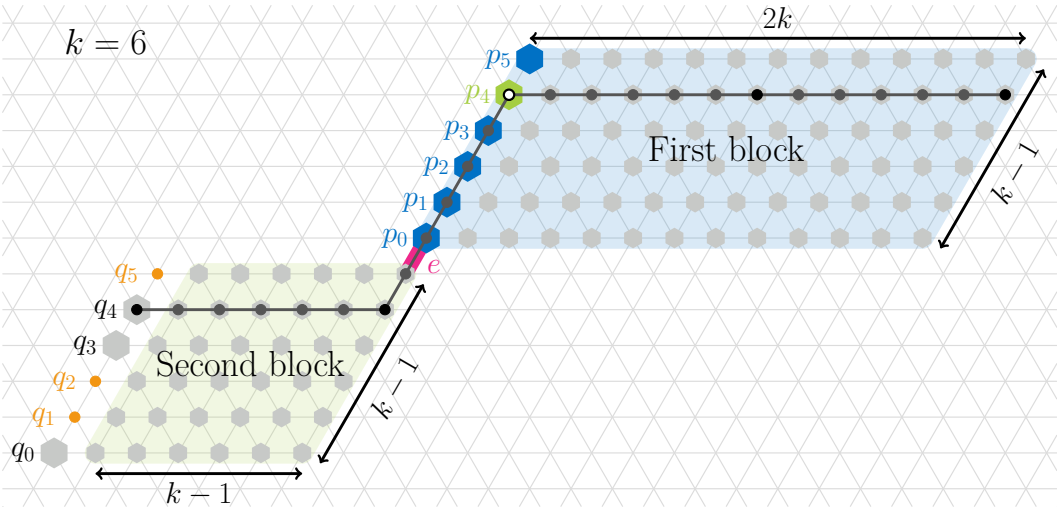


Figure 6 Overview of the amoebot system construction for scale $k = 6$. The first block is shaded blue and the second block is shaded green. The nodes q_i that are not contained in the structure are colored orange. Amoebot p_4 is a valid placement of $k \cdot S$ because q_4 is part of the structure.

24 Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. A survey of autonomous self-reconfiguration methods for robot-based programmable matter. *Robotics and Autonomous Systems*, 120:103242, 2019. doi:10.1016/j.robot.2019.07.012.

25 Tommaso Toffoli and Norman Margolus. Programmable Matter: Concepts and Realization. *International Journal of High Speed Computing*, 05(02):155–170, 1993. doi:10.1142/S0129053393000086.

26 Lidong Yang, Jiangfan Yu, Shihao Yang, Ben Wang, Bradley J. Nelson, and Li Zhang. A Survey on Swarm Microrobotics. *IEEE Transactions on Robotics*, 38(3):1531–1551, 2022. doi:10.1109/TRO.2021.3111788.

A A Lower Bound for Finding Valid Placements

▶ **Theorem 5.** *There exists a shape S_Ω such that for any choice of origin and every amoebot algorithm $\mathcal{A}$ that terminates after $o(\sqrt{n})$ rounds, there exists an amoebot structure A for which the algorithm does not compute $\mathcal{V}(k_{\max}(S_\Omega, A) \cdot S_\Omega, A)$, even if $k_{\max}$ is known.*

Proof. We use the shape S_Ω with a long arm and a short arm connected by a diagonal edge, as depicted in Fig. 4. Let $\mathcal{A}$ be an amoebot algorithm that terminates in $o(\sqrt{n})$ rounds. For every $k \in \mathbb{N}$, we will construct a set A_k of amoebot structures such that $k_{\max}(S_\Omega, A) = k$ for all $A \in A_k$ and only one rotation matches at this scale. Let $k \in \mathbb{N}$ be arbitrary, then we construct A_k as follows (see Fig. 6 for reference):

First, we place a parallelogram of width $2k$ and height $k-1$ with its lower left corner at the origin and call this the *first block*. The first block contains $(2k+1)k = 2k^2 + k$ amoebots and is shared by all $A \in A_k$. Let $p_0, \ldots, p_{k-1}$ be the nodes occupied by the left side of the parallelogram, ordered from bottom to top. Next, we place a second parallelogram with width and height $k-1$ such that its right side extends the first block's left side below the origin. This second block contains k^2 amoebots and is also the same for all structures. It is only connected to the first block by a single edge, e. Let $q_0, \ldots, q_{k-1}$ be the nodes one step to the left of the second block, again ordered from bottom to top.

We define A_k as the set of amoebot structures that consist of these two blocks and m additional amoebots on the positions $q_0, \dots, q_{k-1}$, where $1 \leq m \leq k$. Thus, A_k contains $2^k - 1$ distinct structures. Now, consider placements of S_Ω with maximum scale in any structure $A \in A_k$. For $m = k$, there are exactly k valid placements at scale k, represented by the amoebots $p_0, \dots, p_{k-1}$. The longest continuous lines of amoebots in A have length $2k$ and form the first block. In every valid placement, the longer arm of $k \cdot S_\Omega$ must occupy one of these lines, so no larger scales or other rotations are possible. If q_i is not occupied for some $0 \leq i \leq k-1$, then p_i is not a valid placement because the end of the shorter arm of $k \cdot S_\Omega$ would be placed on q_i. At least one q_i is always occupied, so the maximum scale of S_Ω is k for every $A \in A_k$. Observe that every structure $A \in A_k$ has a unique configuration of valid placements of $k \cdot S_\Omega$: $p \in \mathcal{V}(k \cdot S_\Omega, A)$ if and only if $p = p_i$ and $q_i \in A$ for some $0 \leq i \leq k-1$.

Next, consider the size of the structures in A_k. The maximum number of amoebots is $2k^2 + k + k^2 + k = 3k^2 + 2k$, obtained for $m = k$. This means we have $n \leq 3k^2 + 2k \leq 4k^2$ for large enough k, i.e., $k \geq \sqrt{n}/2$ for all $k \geq 2$ and all $A \in A_k$.

Let $A \in A_k$ be arbitrary and consider the final states of $p_0, \dots, p_{k-1}$ after $\mathcal{A}$ has been executed on A. Each amoebot must be categorized as either a valid or an invalid placement of $k \cdot S_\Omega$. We can assume that this categorization is independent of any randomized decisions because otherwise, there would be a non-zero probability of false categorizations. Thus, the final state depends only on the structure A itself. Recall that structures in A_k only differ in the positions $q_0, \dots, q_{k-1}$ and every path between q_i (or an occupied neighbor) and p_i must traverse the single edge e connecting the two blocks. We can assume that all communication happens via circuits (see Sec. 1.2). Since the first block is the same in all structures, the final states of $p_0, \dots, p_{k-1}$ only depend on the sequence of signals sent from the second block to the first block through e. To compute the correct set of valid placements, each amoebot structure in A_k therefore has to produce a unique sequence of signals: If for any two configurations, the same sequence of signals is sent through e, the final states of $p_0, \dots, p_{k-1}$ will be identical, so at least one will be categorized incorrectly.

Let c_ϕ be the number of pins used by $\mathcal{A}$. Then, the number of different signals that can be sent via one edge in one round is $2^{c_\phi} = \mathcal{O}(1)$, and the number of signal sequences that can be sent in r rounds is 2^{rc_ϕ}. Therefore, to produce at least $2^k - 1$ different sequences of signals, we require $r = \Omega(k/c_\phi) = \Omega(\sqrt{n})$ rounds. By the assumption that $\mathcal{A}$ terminates after $o(\sqrt{n})$ rounds, $\mathcal{A}$ will produce at least one false result for sufficiently large k.

It remains to be shown that the same arguments hold for all equivalent versions of S_Ω that contain the origin. If the origin is placed on another node of the longer arm, the valid placement candidates $p_0, \dots, p_{k-1}$ are shifted to the right by k or $2k$ steps, respectively; everything else remains the same. If the origin is placed on the shorter arm of the shape, we switch the roles of the first and the second block. We place amoebots on all positions $q_0, \dots, q_{k-1}$ and use the right side of the first block as the controlling positions instead. The number and size of the resulting amoebot structures remain the same, so the same arguments hold as before. ◀

B Self-Contained Shapes are Star Convex

To show the properties of star convex shapes, we will use the following equivalent characterization:

▶ **Lemma 24.** *A shape S is star convex with its origin as a center node if and only if S is the union of parallelograms of the form $\mathrm{L}(d, \ell) \oplus \mathrm{L}(d', \ell')$ and convex shapes of the form $\mathrm{T}(d, 1) \oplus \mathrm{L}(d, \ell) \oplus \mathrm{L}(d', \ell')$, where d' is obtained from d by a $60°$ clockwise rotation. The number of these shapes is linear in $|V(S)|$.*

For the proof of Theorem 21, we first show several lemmas that provide the necessary tools. To start with, we show that non-star convex shapes cannot become star convex by scaling, and for a sufficiently large scale, every center candidate has a shortest path with a missing edge. It is clear that star convex shapes stay star convex after scaling (by Lemma 24), but non-star convex shapes gain new potential center nodes, making it less obvious why there still cannot be a center.

▶ **Lemma 25.** *Let S be an arbitrary non-star convex shape, then for every $k \in \mathbb{N}$, $k \cdot S$ is not star convex, and for $k > 2$ and every node $c \in V(k \cdot S)$, there exists a shortest path from c to a node $v \in V(k \cdot S)$ in G_Δ with at least one edge not contained in $k \cdot S$.*

Next, we show a *fixed point* property that is particularly useful for scales k and $k+1$.

▶ **Lemma 26.** *Let S be a shape and $k \in \mathbb{N}$ a scale such that there exists a $t \in V_\Delta$ with $k \cdot S + t \subseteq (k+1) \cdot S$. Then, t must be in $V(S)$ and we call t a* fixed node *of S. Further, for every node $v \in V(S)$, a shortest path from $k \cdot v + t$ to $(k+1) \cdot v$ is also a shortest path from t to v and vice versa.*

Finally, we eliminate the need for covering rotations by showing that for sufficiently large scales k and $k+1$, $k \cdot S^{(r)}$ does not fit into $(k+1) \cdot S$ for any $r \in \{1, \dots, 5\}$ unless S is rotationally symmetric.

▶ **Definition 27.** *A shape S is called* rotationally symmetric with respect to $r \in \{1, 2, 3\}$ *(or* r-symmetric*) if there exists a translation $t \in V_\Delta$ such that $S^{(r)} + t = S$.*

Note that $r \in \{1, 2, 3\}$ covers all possible rotational symmetries in the triangular grid and 1-symmetry is equivalent to 2- and 3-symmetry combined. Additionally, the translation t is unique. Also note that 1-symmetry is more commonly called 6-*fold* symmetry, 2-symmetry is known as 3-fold symmetry and 3-symmetry is known as 2-fold symmetry.

▶ **Lemma 28.** *Let S be a shape and $r \in \{1, 2, 3\}$ be arbitrary. If S is not r-symmetric, then there is a scale $k_0 \in \mathbb{N}$ such that for every $k \geq k_0$, $k \cdot S^{(r)}$ does not fit into $(k+1) \cdot S$.*

Using these lemmas, we can now prove the main theorem about self-contained and star convex shapes:

▶ **Theorem 21.** *A shape is self-contained if and only if it is star convex.*

Proof. First, let S be star convex with center node c and consider two scale factors $k < k'$. By Lemma 24, S can be represented as

$$S = \left(\bigcup_{i=1}^{m_1} P_i \cup \bigcup_{j=1}^{m_2} T_j \right) + c,$$

where the P_i are parallelograms and the T_i are Minkowski sums of parallelograms with triangles. Then, we have $k \cdot P_i \subseteq k' \cdot P_i$ and $k \cdot T_j \subseteq k' \cdot T_j$ since each of the P_i and T_j is a convex shape. We choose t such that $k \cdot c + t = k' \cdot c$, then each of the constituent shapes of $k \cdot S + t$ is contained in its counterpart in $k' \cdot S$. This already shows that every star convex shape is self-contained.

Now, let S be a shape that is not star convex. We will show that S is not self-contained by finding a scale $k \in \mathbb{N}$ such that for every $t \in \mathbb{R}^2$ and $r \in \{0, \dots, 5\}$, $(k \cdot S^{(r)} + t) \setminus ((k+1) \cdot S) \neq \emptyset$. Using Lemma 25, we can assume that for every node $c \in V(S)$, there exists a shortest path Π to a node $v \in V(S)$ such that at least one edge of Π is not contained in S. If this is not the case for S already, we simply consider $3 \cdot S$ as the new base shape.

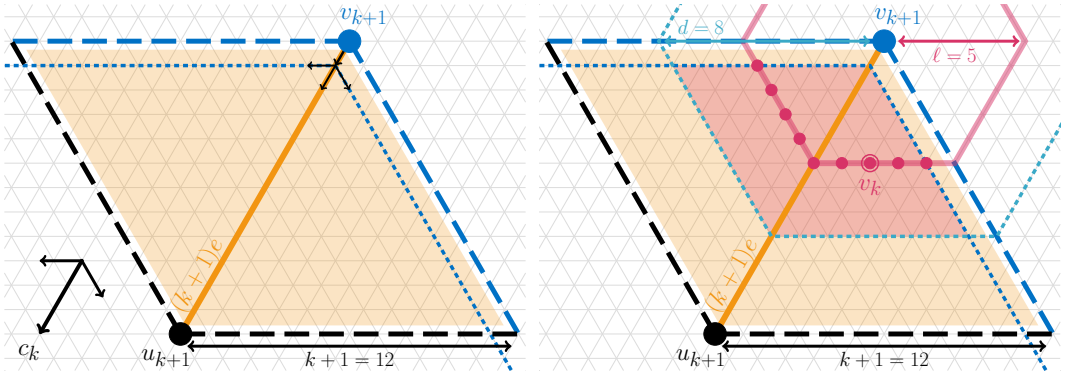


Figure 7 Illustration of the constraints forcing v_k to lie in a region unoccupied by $(k+1)\cdot S$ for $k=11$. The scaled unoccupied edge $(k+1)\cdot e$ and the empty parallelogram region it produces are highlighted and shaded in orange. The parallelogram's edges are drawn as dashed lines of length $k+1$, incident to u_{k+1} and v_{k+1}. At most these edges of the parallelogram can be occupied by $(k+1)\cdot S$. The distance between c and v resp. v_k and v_{k+1} is $\ell=5<k+1$. In the bottom left corner of (a), the directions used by all shortest paths from v_{k+1} to v_k (resp. v to c) are shown, with the opposite direction of e emphasized because at least one such edge must be on every path. This prevents v_k from lying on an edge incident to v_{k+1}, shown by the dotted blue lines and the small black arrows. Similarly, v_k cannot lie on any of the dashed black edges incident to u_{k+1} because it must be closer to v_{k+1}. In particular, its distance to v_{k+1} is bounded by $d=8$, as indicated by the light blue dotted lines in (b). The red lines show the nodes with distance exactly ℓ to v_{k+1}. By combining all constraints, v_k has to be one of the red nodes in the dark shaded area, none of which are occupied by $(k+1)\cdot S$.

By Lemma 28, we can find k_0 large enough that no non-zero rotation of $k\cdot S$ fits into $(k+1)\cdot S$ for any scale $k\geq k_0$ unless S is rotationally symmetric. In this case, however, the rotated version of S is the same as a translation of S, so we can disregard rotations altogether because they do not affect the existence of valid translations of $k\cdot S$ in $(k+1)\cdot S$.

Let H be the convex hull of S and let $d\in\mathbb{N}$ be its diameter. Consider any scale $k>d$ and some placement $c\in V_\Delta$ such that $k\cdot S+c\subseteq(k+1)\cdot S$. By Lemma 26, $c\in V(S)$. As argued above, there is a node $v\in V(S)$ such that a shortest path Π from c to v has at least one edge that is not contained in S. We choose v and Π such that the last edge is missing, w.l.o.g. Let $u\in V(H)$ be the predecessor of v on Π, i.e., the missing edge e of Π connects u and v. Since S does not contain e, it cannot contain the two incident faces. Thus, there is an area in the shape of a regular parallelogram with side length $k+1$ whose diagonal is $(k+1)\cdot e$ and whose interior does not intersect $(k+1)\cdot S$, i.e., only its edges could be covered by edges of $(k+1)\cdot S$ (see Fig. 7). These edges are incident to $v_{k+1}=(k+1)\cdot v$ and $u_{k+1}=(k+1)\cdot u$ and they lie on different axes than e. By Lemma 26, the grid distance between $v_k=k\cdot v+c$ and v_{k+1} is the length of Π, which is bounded by d. By the choice of $k>d$, the distance between u_{k+1} and v_{k+1} (which is $k+1$) is therefore greater than the distance between v_k and v_{k+1}. Now, observe that for any node on one of the parallelogram's edges incident to u_{k+1}, the distance to v_{k+1} remains $k+1$. Thus, v_k cannot lie on any of these two edges.

Furthermore, recall from Lemma 26 that any shortest path from c to v is also a (translated) shortest path from v_k to v_{k+1}. Because such a path contains at least one edge parallel to e, every shortest path from v_{k+1} to v_k contains at least one edge in the opposite direction

of e. Therefore, since the edges of the empty parallelogram that are incident to v_{k+1} lie on different axes than e, v_k cannot lie on these two edges either. The direction of e also implies that v_k must be on the side of the two edges that is closer to u_{k+1}.

Together, these constraints imply that v_k lies *inside* the parallelogram that is not contained in $(k+1)\cdot S$, so the placement identified by c does not satisfy $k\cdot S+c \subseteq (k+1)\cdot S$, contradicting our assumption. Since this works for every choice of $c \in V(S)$, there is no placement of $k \cdot S$ in $(k+1) \cdot S$. ◀

C Main Theorem

▶ **Theorem 23.** *Let A be an amoebot structure and S a snowflake shape. Given a tree representation of S, the amoebots can compute $k = k_{\max}(S, A)$ in a binary counter and determine $\mathcal{V}(k \cdot S^{(r)}, A)$ for all $r \in \{0, \dots, 5\}$ within $\mathcal{O}(\log^2 k)$ rounds if S is star convex and $\mathcal{O}(K \log K)$ rounds otherwise, where $k \leq K = k_{\max}(\mathrm{T}(E, 1), A) = \mathcal{O}(\sqrt{n})$.*

Proof. The amoebots first establish binary counters on all maximal segments in A, for all axes. At least one of these will be large enough to store $k = k_{\max}(S, A)$ as long as S is non-trivial. In the following, they use all these counters simultaneously and deactivate every counter exceeding its memory during an operation.

Consider the case where S is star convex. By Lemma 8, combined with Lemma 19, the amoebots can compute $k_{\max}$ using a binary search and find all valid placements at all six rotations within $\mathcal{O}(\log^2 k)$ rounds. Now, let S be a snowflake that is not star convex. In this case, S must contain at least one triangular face because all snowflakes without faces are unions of lines meeting at the origin, which are star convex (observe that the Minkowski sum of an edge with a line on a different axis always contains some faces). Then, $K = k_{\max}(\mathrm{T}(\mathrm{E}, 1), A)$ is an upper bound for $k_{\max}(S, A)$. The amoebots can compute K within $\mathcal{O}(\log^2 K)$ rounds and store it in binary counters, since triangles are star convex. A simple linear search for $k_{\max}$ yields the runtime of $\mathcal{O}(K \log K)$ by Lemma 6. $K = \mathcal{O}(\sqrt{n})$ follows from the fact that the number of nodes covered by $k \cdot \mathrm{T}(\mathrm{E}, 1)$ grows quadratically with k. ◀