Natural Computing
https://doi.org/10.1007/s11047-024-09981-6

The structural power of reconfigurable circuits in the amoebot model

Andreas Padalkin[1] · Christian Scheideler[1] · Daniel Warner[1]

Accepted: 29 February 2024

Abstract

The *amoebot model* (Derakhshandeh et al. in: SPAA ACM, pp 220–222. https://doi.org/10.1145/2612669.2612712, 2014) has been proposed as a model for programmable matter consisting of tiny, robotic elements called *amoebots*. We consider the *reconfigurable circuit extension* (Feldmann et al. in J Comput Biol 29(4):317–343. https://doi.org/10.1089/cmb.2021.0363, 2022) of the geometric amoebot model that allows the amoebot structure to interconnect amoebots by so-called *circuits*. A circuit permits the instantaneous transmission of signals between the connected amoebots. In this paper, we examine the structural power of the reconfigurable circuits. We start with fundamental problems like the *stripe computation problem* where, given any connected amoebot structure S, an amoebot u in S, and some axis X, all amoebots belonging to axis X through u have to be identified. Second, we consider the *global maximum problem*, which identifies an amoebot at the highest possible position with respect to some direction in some given amoebot (sub)structure. A solution to this problem can be used to solve the *skeleton problem*, where a cycle of amoebots has to be found in the given amoebot structure which contains all boundary amoebots. A canonical solution to that problem can be used to come up with a canonical path, which provides a unique characterization of the shape of the given amoebot structure. Constructing canonical paths for different directions allows the amoebots to set up a spanning tree and to check symmetry properties of the given amoebot structure. The problems are important for a number of applications like rapid shape transformation, energy dissemination, and structural monitoring. Interestingly, the reconfigurable circuit extension allows polylogarithmic-time solutions to all of these problems.

Keywords Progammable matter · Amoebot model · Reconfigurable circuits · Spanning tree · Symmetry detection

1 Introduction

The *amoebot model* (Derakhshandeh et al. 2014; Daymude et al. 2023) is a well-studied model for programmable matter (Toffoli and Margolus 1993)—a substance that can be programmed to change its physical properties, like its shape and density. In the geometric variant of this model, the substance (called the *amoebot structure*) consists of simple particles (called *amoebots*) that are placed on the infinite triangular grid graph and are capable of local movements through *expansions* and *contractions*.

Inspired by the *nervous* and *muscular system*, Feldmann et al. (2022) introduced a *reconfigurable circuit extension* to the amoebot model with the goal of significantly accelerating fundamental problems like leader election and shape transformation. As a first step, they showed that leader election, consensus, compass alignment, chirality agreement, and various shape recognition problems can be solved in at most $O(\log n)$ time. This paper continues this line of work by considering a number of additional problems:

First, we consider the *stripe computation problem* where, given any connected amoebot structure S, an amoebot u in S, and some axis X, all amoebots belonging to axis X through u have to be identified. Second, we consider the *global maximum problem*, which identifies an amoebot at the highest possible position with respect to some direction in some given amoebot (sub)structure. A solution

✉ Andreas Padalkin
andreas.padalkin@upb.de

Christian Scheideler
scheideler@upb.de

Daniel Warner
dwarner@upb.de

[1] Department of Computer Science, Paderborn University, Fürstenallee 11, 33102 Paderborn, Germany

Published online: 13 April 2024

to this problem can then be used to solve the *skeleton problem*, where a (not necessarily simple) cycle of amoebots has to be found in the given amoebot structure which contains all boundary amoebots. A canonical solution to that problem can then be used to come up with a canonical path, which provides a unique characterization of the shape of the given amoebot structure. Constructing canonical paths for different directions will then allow the amoebots to set up a spanning tree and to check symmetry properties of the given amoebot structure.

The problems have a number of important applications. The stripe computation problem is important to avoid conflicts in joint amoebot contractions and expansions (see Figs. 1 and 2), which is critical for rapid shape transformation. A spanning tree is an important step towards energy distribution from amoebots with access to energy to amoebots without such access (Daymude et al. 2021), and canonical skeleton paths as well as symmetry checks are important for structural monitoring and repair.

1.1 Geometric amoebot model

In the *geometric amoebot model* (Daymude et al. 2023), a set of n amoebots is placed on the infinite regular triangular grid graph $G_\Delta = (V, E)$ (see left side of Fig. 3). An amoebot is an anonymous, randomized finite state machine that either occupies one or two adjacent nodes of G_Δ, and every node of G_Δ is occupied by at most one amoebot. If an amoebot occupies just one node, it is called *contracted* and otherwise *expanded*. Two amoebots that occupy adjacent nodes in G_Δ are called *neighbors*. Amoebots are able to move through *contractions* and *expansions*. However, since our algorithms do not make use of movements, we omit further details and refer to Daymude et al. (2023) for more information.

Each amoebot has a compass orientation (it defines one of its incident edges as the northern direction) and a chirality (a sense of clockwise or counterclockwise rotation)

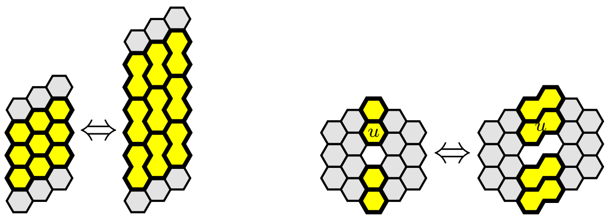


Fig. 1 Joint movement extension. Feldmann et al. (2022) have proposed joint movements to the amoebot model where an expanding amoebot is capable of pushing other amoebots away from it, and a contracting amoebot is capable of pulling other amoebots towards it. The figures show a joint expansion (from left to right) resp. a joint contraction (from right to left) of the yellow amoebots (thick boundary). The left side of the right figure shows stripe $\mathrm{A}(S, u, N)$ (see Sect. 1.3). The stripe can expand without causing any conflicts

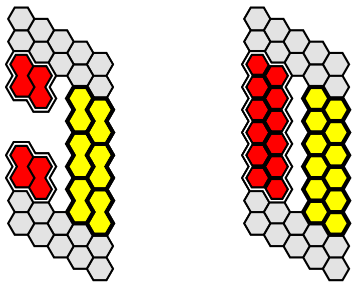


Fig. 2 Exemplary conflicts. In the left figure, if the yellow amoebots (thick boundary) contract, the red amoebots (double boundary) will collide. In order to avoid the collision, the red amoebots (double boundary) have to contract as well. In the right figure, if the yellow amoebots (thick boundary) expand, the amoebot structure will tear apart. In order to maintain all connections, the red amoebots (double boundary) have to expand as well

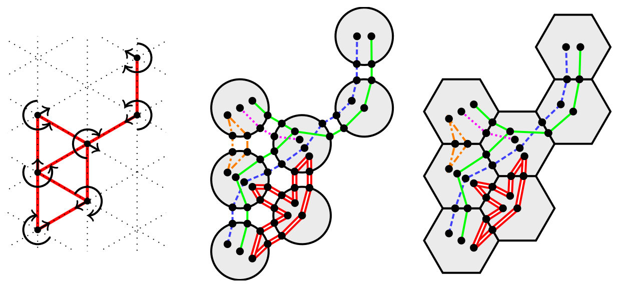


Fig. 3 Amoebot structures and reconfigurable circuits. The first figure shows an amoebot structure S. The dotted lines indicate the triangular grid G_Δ. The nodes indicate the amoebots. The arrows show their chirality and compass orientation. The red edges indicate the graph G_S. The other two figures show an amoebot structure with $k = 2$ external links between neighboring amoebots. The amoebots are shown in gray. The nodes on the boundary are the pins, and the ones within the amoebots the partition sets. An edge between a partition set Q and a pin p implies $p \in Q$. Each color resp. line pattern indicates another circuit. The first two figures are taken from Feldmann et al. (2022)

that it can maintain as it moves, but initially the amoebots might not agree on their compass orientation and chirality. In this paper, we assume that all amoebots share a common compass orientation and chirality. This is reasonable since Feldmann et al. (2022) showed that all amoebots are able to come to an agreement within the considered extension (see Sect. 1.4).

Let the *amoebot structure* $S \subseteq V$ be the set of nodes occupied by the amoebots. By abuse of notation, we identify amoebots with their nodes. We say that S is *connected* iff G_S is connected, where $G_S = G_\Delta|_S$ is the graph induced by S. In this paper, we assume that initially, S is connected and all amoebots are contracted. Also, we assume the fully synchronous activation model, i.e., the time is divided into synchronous rounds, and every amoebot is active in each round. On activation, each amoebot may perform a movement and update its state as a function of its previous state. However, if an amoebot fails to perform its movement, it remains in its previous state. The time complexity of an algorithm is measured by the number of synchronized rounds required by it.

1.2 Reconfigurable circuit extension

In the *reconfigurable circuit extension* (Feldmann et al. 2022), each edge between two neighboring amoebots u and v is replaced by k edges called *external links* with endpoints called *pins*, for some constant $k \geq 1$ that is the same for all amoebots. For each of these links, one pin is owned by u while the other pin is owned by v. In this paper, we assume that neighboring amoebots have a common labeling of their incident external links.

Each amoebot u *partitions* its *pin set* $P(u)$ into a collection $\mathcal{Q}(u)$ of pairwise disjoint subsets such that the union equals the pin set, i.e., $P(u) = \bigcup_{Q \in \mathcal{Q}(u)} Q$. We call $\mathcal{Q}(u)$ the *pin configuration* of u and $Q \in \mathcal{Q}(u)$ a *partition set* of u. Let $\mathcal{Q} = \bigcup_{u \in S} \mathcal{Q}(u)$ be the collection of all partition sets in the system. Two partition sets are *connected* iff there is at least one external link between those sets. Let L be the set of all connections between the partition sets in the system. Then, we call $H = (\mathcal{Q}, L)$ the *pin configuration* of the system and any connected component C of H a *circuit* (see center of Fig. 3). Note that if each partition set of $\mathcal{Q}$ is a *singleton*, i.e., a set with exactly one element, then every circuit of H just connects two neighboring amoebots. However, an external link between the neighboring amoebots u and v can only be maintained as long as both u and v occupy the incident nodes. Whenever two amoebots disconnect, the corresponding external links and their pins are removed from the system. An amoebot is part of a circuit iff the circuit contains at least one of its partition sets. A priori, an amoebot u may not know whether two of its partition sets belong to the same circuit or not since initially it only knows $\mathcal{Q}(u)$.

Each amoebot u can send a primitive signal (a *beep*) via any of its partition sets $Q \in \mathcal{Q}(u)$ that is received by all partition sets of the circuit containing Q at the beginning of the next round. The amoebots are able to distinguish between beeps arriving at different partition sets. More specifically, an amoebot receives a beep at partition set Q if at least one amoebot sends a beep on the circuit belonging to Q, but the amoebots neither know the origin of the signal nor the number of origins. Note that beeps are enough to send whole messages over time, especially between adjacent amoebots. We modify an activation of an amoebot as follows. As a function of its previous state and the beeps received in the previous round, each amoebot may perform a movement, update its state, reconfigure its pin configuration, and activate an arbitrary number of its partition sets. The beeps are propagated on the updated pin configurations. If an amoebot fails to perform its movement, it remains in its previous state and pin configuration, and does not beep on any of its partition sets.

In this paper, we will utilize the dual graph of the triangular grid graph, i.e., a hexagonal tesselation, to visualize amoebot structures (see right side of Fig. 3). Thereby, we reduce each external link to a single pin. Furthermore, in order to improve the comparability of circuit configurations, we add pins to each side of the hexagon.

1.3 Problem statement and our contribution

Let $D_m = \{N, ENE, ESE, S, WSW, WNW\}$ be the set of all cardinal directions along the axes of G_Δ, and $D_p = \{E, SSE, SSW, W, NNW, NNE\}$ the set of all cardinal directions perpendicular to the axes of G_Δ (see Fig. 4). In the following, we state the considered problems. An overview is given by Table 1.

First, we consider the *stripe computation problem*. Let $\mathrm{X}(v, d) \subseteq V$ denote the nodes of G_Δ that lie on the axis through the node $v \in V$ into the cardinal direction $d \in D_m \cup D_p$. For $R \subseteq V$, we call the set $\mathrm{A}(R, v, d) = R \cap \mathrm{X}(v, d)$ a *stripe* of R (see Fig. 1). Note that a stripe is not necessarily connected. Let an amoebot $u \in S$ and a cardinal direction $d \in D_m \cup D_p$ be given, i.e., each amoebot $v \in S$ knows the cardinal direction d and whether $v = u$. The goal of each amoebot $v \in S$ is to determine whether $v \in \mathrm{A}(S, u, d)$. Our *stripe algorithm* solves the stripe computation problem after $O(\log n)$ rounds.

Second, we consider the *global maxima problem*. Let a cardinal direction $d \in D_m \cup D_p$ and a non-empty set $R \subseteq S$ be given, i.e., each amoebot $v \in S$ knows the direction d and whether $v \in R$. The goal of each amoebot $v \in S$ is to determine whether $v \in \operatorname{argmin}_{w \in R} \mathrm{f}_d(R, w)$ where $\mathrm{f}_d(R, w)$ denotes the number of amoebots in R that lie in direction d from amoebot w. We call $\operatorname{argmin}_{w \in R} \mathrm{f}_d(R, w)$ the set of global maxima of R with respect to d. Our *global maxima algorithm* solves the global maxima problem after $O(\log^2 n)$ rounds w.h.p.[1]

Third, we consider the *(canonical) skeleton problem*. An amoebot u is a *boundary amoebot* iff it is adjacent to an unoccupied node in $V \setminus S$. Otherwise, we call u an *inner amoebot*. A (potentially non-simple) cycle of amoebots is a skeleton iff the cycle contains all boundary amoebots in

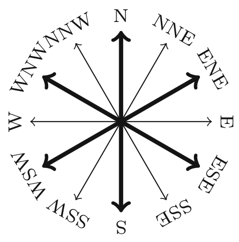


Fig. 4 Cardinal directions. The thick arrows indicate the cardinal directions along the main axes (D_m), and the thin ones the cardinal directions perpendicular to the main axes (D_p)

Table 1 An overview of our main algorithmic results

Problem	Required pins	Runtime	Section	Theorem
Stripe computation	2	$O(\log n)$	Section 3.3	Theorem 3.7
Global maxima	2	$O(\log^2 n)$ w.h.p.	Section 3.3	Theorem 3.9
Canonical skeleton	4	$O(\log^2 n)$ w.h.p.	Section 4.1	Theorem 4.4
Spanning tree	4	$O(\log^2 n)$ w.h.p.	Section 4.2	Theorem 4.6
Symmetry detection	4	$O(\log^5 n)$ w.h.p.	Section 4.3	Theorem 4.9

S. Note that the skeleton may contain inner amoebots. An amoebot structure computes a skeleton C iff each amoebot knows its predecessor and successor for each of its occurrences in C. The goal of the skeleton problem is to compute an arbitrary skeleton.

Since skeletons are not unique, we define a *canonical skeleton* with respect to a cardinal direction $d \in D_m \cup D_p$ and a sign $s \in \{+, -\}$ (abbreviated as (d, s)-skeleton). We defer the definition to Sect. 4.1. Let a cardinal direction $d \in D_m \cup D_p$ and a sign s be given, i.e., each amoebot $v \in S$ knows the cardinal direction d and the sign s. The goal of the canonical skeleton problem is to compute the canonical skeleton. Our *canonical skeleton algorithm* solves the (canonical) skeleton problem after $O(\log^2 n)$ rounds w.h.p.

Our algorithms for the remaining problems are based on skeletons. However, they split the skeletons into paths that we call *skeleton paths*. For the canonical skeletons, we define a *canonical skeleton path* by specifying a splitting point. Our canonical skeleton algorithm determines this point in parallel to the computation of the canonical skeleton.

Fourth, we consider the *spanning tree problem*. A *tree* is a cycle-free and connected graph. A *spanning tree* of an amoebot structure S is a tree $T = (S, E_T)$ with $E_T \subseteq E$. An amoebot structure computes a spanning tree T if each amoebot $u \in S$ knows whether $\{u, v\} \in E_T$ for each neighbor $v \in \mathrm{N}(u)$. The goal of the spanning tree problem is to compute an arbitrary spanning tree. Our *spanning tree algorithm* solves the spanning tree problem after $O(\log^2 n)$ rounds w.h.p.

Finally, we consider the *symmetry detection problem*. The goal of that problem is to determine whether the amoebot structure features rotational or reflection symmetries. Our *symmetry detection algorithm* solves the the problem after $O(\log^5 n)$ rounds w.h.p. In addition, we show how to compute the amoebot occupying the symmetry point and amoebots on the symmetry axis, respectively, if such exist. Our procedures require $O(\log^2 n)$ rounds w.h.p.

Note that our stripe algorithm is deterministic while all other algorithms are randomized. The global maxima, canonical skeleton, and spanning tree algorithm only require randomization for leader election (see Thereom 2.1).

1.4 Related work

The reconfigurable circuit extension was introduced by Feldmann et al. (2022). Note that the amoebot model (Alumbaugh et al. 2019) and its circuit extension (Padilla et al. 2015; Scalise and Schulman 2019; Shah et al. 2020; Song et al. 2019) can in principle be realized.

Feldmann et al. (2022) have proposed solutions for *leader election* (see Sect. 2), *consensus*, *compass alignment*, *chirality agreement*, and various *shape recognition* problems. Both, the alignment of the compasses and the agreement on a chirality requires $O(\log n)$ w.h.p. This makes our assumption of a common compass orientation and chirality reasonable.

To our knowledge the stripe computation, global maxima, (canonical) skeleton, and symmetry detection problem have not been considered within the standard amoebot model. However, regarding the global maxima problem, Daymude et al. (2020) have considered the related problem of determining the dimensions of an object (a finite, connected, static set of nodes) in order to solve various *convex hull problems*. Their approach can be easily adjusted to compute the global maxima of the amoebot structure. However, it requires $O(n)$ rounds.

The spanning tree problem is widely studied in the distributed algorithms community, e.g., Pandurangan et al. (2018). The *spanning tree primitive* is one of the most used techniques to move amoebots, e.g., Daymude et al. (2019), Daymude et al. (2020), Derakhshandeh et al. (2015), Luna et al. (2020). Beyond that, spanning trees were applied to distribute energy (Daymude et al. 2021). However, since in the standard amoebot model, information can only travel amoebot by amoebot, the construction requires $\Omega(D)$ rounds where D is the diameter of the amoebot structure

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

(e.g., see Daymude et al. (2019)). Since $D = \Omega(\sqrt{n})$, our solution is a significant improvement.

Other problems that were considered in the standard amoebot model include shape formation (e.g., see Derakhshandeh et al. 2016; Luna et al. 2020), object coating (Derakhshandeh et al. 2017), convex hull (Daymude et al. 2020), gathering (Cannon et al. 2016), and bridging (Arroyo et al. 2018). It would certainly be interesting to investigate whether the reconfigurable circuit extension can also be used to significantly speed up solutions to these problems, but we leave it as future work.

2 Preliminaries

In this section, we enumerate important primitives given in previous papers.

Global circuit: If each amoebot partitions its pins into one partition set, we obtain a single circuit that interconnects all amoebots. We call this circuit the *global circuit* (Feldmann et al. 2022).

Leader election: We make use of a generalized version of the leader election algorithm proposed by Feldmann et al. (2022):

Theorem 2.1 *Let $C_1, \ldots, C_m$ be sets of candidates. For each $i \in \{1, \ldots, m\}$, let $\mathcal{C}_i$ be the circuit that connects all candidates of set C_i. Let $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ hold for all $i \neq j$. An amoebot structure elects a leader from each set of candidates after $\Theta(\log n)$ rounds w.h.p.*

In the classical leader election problem, the amoebot structure S has to elect a leader only from the set $C_1 = S$.

Chains: We call an ordered sequence $C = (u_0, \ldots, u_{m-1})$ of m amoebots a *chain* iff (i) all subsequent amoebots u_i, u_{i+1} are neighbors, (ii) each amoebot in C except u_0 knows its predecessor, and (iii) each amoebot in C except u_{m-1} knows its successor.

Boundary sets: We adopt the definition for the *boundary sets* from Derakhshandeh et al. (2015): Consider $G_{V \setminus S} = G_\Delta|_{V \setminus S}$. The connected components of $G_{V \setminus S}$ are called empty regions. The number of empty regions is finite since S is finite. Let $R_1, \ldots, R_m$ denote the empty regions. For $i \in \{1, \ldots, m\}$, the boundary set B_i is the neighborhood of R_i in S, i.e.,

$$B_i = \{u \in S \mid \exists v \in R_i : \{u, v\} \in E\}.$$

There is exactly one infinite empty region since S is finite. We call the corresponding boundary set the *outer boundary set*, and the others *inner boundary sets*.

Derakhshandeh et al. (2015) showed that the amoebot structure is able to organize each boundary set into a cycle. For that, each amoebot $u \in S$ proceeds as follows. Consider the intersection of $G_{V \setminus S}$ and the neighborhood of u. For each connected component in the intersection, amoebot u can identify the next clockwise neighbor $v \in S$ and the next counterclockwise neighbor $w \in S$. Amoebot u picks v as its successor, and w as its predecessor. Since we assume that the amoebots agree on a common chirality, we obtain a cycle for each boundary set.

In order to distinguish inner and outer boundaries, we apply the *inner outer boundary test* by Derakhshandeh et al. (2015). They accumulate the angles of the turns while traversing the cycle of the boundary set once. An outer boundary set results in a sum of 360°, and an inner boundary set results in a value of −360°. Note that it is sufficient to count the turns by 60° modulo 5. This allows us to accumulate the sum with constant memory. However, the traversing requires $O(n)$ rounds. We accelerate the accumulation by the following result by Feldmann et al. (2022):

Theorem 2.2 *Let $C = (v_0, \ldots, v_{m-1})$ be a chain within the amoebot structure where $m \in \mathbb{N}$ denotes the length of the chain. Let $k \in \mathbb{N}$ be constant. Let $x_i \in \{0, \ldots, k-1\}$ for all $i \in \{0, \ldots, m-1\}$. Suppose that for each $i \in \{0, \ldots, m-1\}$, amoebot v_i knows the value x_i. Then, the chain computes $x = \sum_{i \in \{0, \ldots, m-1\}} x_i \bmod k$ after $O(\log m)$ rounds.*

Corollary 2.3 *A boundary set can determine whether it is an inner boundary set or the outer boundary set within $O(\log n)$ rounds w.h.p.*

Proof In order to accelerate the inner outer boundary test, we apply Theorem 2.2. Recall that each boundary set can organize itself into a cycle. We apply Theorem 2.1 on each cycle to elect a leader. However, each amoebot operates each of its positions independently of each other. We split each cycle at the elected position. Each amoebot knows its predecessor and successor on the cycle and therefore also within the chain. Let x_i denote the angle at amoebot v_i and let $k = 5$. Finally, note that each boundary has $O(n)$ amoebots since each amoebot has at most three local boundaries (e.g., consider the yellow amoebot labelled with a 0 on the right side of Fig. 14). Thus, the sum of the angles can be accumulated after $O(\log n)$ rounds. □

Synchronization of procedures: Sometimes, we want to execute a procedure on different subsets in parallel, e.g., we apply the inner outer boundary test on all boundary sets at once. The execution of the the same procedure may take different amounts of rounds for each subset. The amoebots of a single subset are not able to decide when all subsets have terminated. In order to synchronize the amoebot structure with respect to the procedures, the amoebots periodically establish the global circuit. The amoebots of each subset that has not yet terminated beep on that circuit.

The amoebot structure may proceed to the next procedure once the global circuit was not activated.

3 Computing identifiers

In this section, we assign *identifiers* in $\mathbb{Z}$ to the amoebots. Let $(x_{k-1}, \ldots, x_0)$ denote the *two's complement representation* of $-x_{k-1} \cdot 2^{k-1} + \sum_{i=0}^{k-2} x_i \cdot 2^i$. By abuse of notation, we identify the identifiers with their two's complement representation. Due to the constant-sized memory of the amoebots, each amoebot has to compute a two's complement representation of its identifier over time, i.e., it will compute the i-th bit in the i-th iteration of the presented algorithms.

In the first subsection, we compute successive identifiers along chains. In the second subsection, we compute spatial identifiers with respect to a cardinal direction. In the third subsection, we show two applications for the identifiers. In the fourth subsection, we compute identifiers equal to the distances to a given amoebot.

3.1 Successive identifiers along the chain

In this section, we compute successive identifiers along a chain with respect to an amoebot that we call the *reference amoebot*. Let $C = (u_0, \ldots, u_{m-1})$ be a chain of amoebots. Let u_r be an arbitrary reference amoebot within the chain, e.g., chosen by position (for example $r = 0$), or by a leader election. We assign identifiers id_{C,u_r} according to the following rules:

$$\mathrm{id}_{C,u_r}(u_r) = 0 \tag{1}$$

$$\mathrm{id}_{C,u_r}(u_{i+1}) = \mathrm{id}_{C,u_r}(u_i) + 1 \text{ for } 1 \le i < m \tag{2}$$

Note that $\mathrm{id}_{C,u_r}(u_i) = i - r$. Also note that the identifiers might be negative.

In order to compute the identifiers, we utilize a procedure on the chain of amoebots proposed by Feldmann et al. (2022) that we henceforth refer to as the *primary and secondary circuit algorithm* (PASC algorithm). Originally, the algorithm has been used as a subroutine for Theorem 2.2.

In the following, we explain how the PASC algorithm works (see Fig. 5). So, let $C = (u_0, \ldots, u_{m-1})$ be a chain of m amoebots. If an amoebot occurs multiple (but constantly many) times, it operates each position independently of each other which is possible by using sufficiently many pins. Each amoebot is either *active* or *passive*. Initially, each amoebot is active. The algorithm iteratively transforms active amoebots into passive amoebots while keeping amoebot u_r active.

At the beginning of an iteration, the amoebots establish two circuits as follows. Each amoebot has two partition sets that we call the *primary and secondary partition set*. We connect the primary and secondary partition sets by the following two rules[2] (see Fig. 5): (i) If an amoebot is active, we connect its primary partition set to the secondary partition set of its predecessor, and its secondary partition set to the primary partition set of its predecessor. (ii) If an amoebot is passive, we connect its primary partition set to the primary partition set of its predecessor, and its secondary partition set to the secondary partition set of its predecessor.

We obtain two circuits through all amoebots (see Fig. 5). Each amoebot defines the circuit containing its primary partition set as its *primary circuit*, and the circuit containing its secondary partition set as its *secondary circuit*. Clearly, we obtain two disjoint circuits along the chain. We refer to Feldmann et al. (2022) for a detailed construction.

After establishing these circuits, the iteration utilizes two rounds. In the first round, amoebot u_r activates its primary circuit. Each active amoebot that has received the beep on its secondary circuit beeps in the second round on its secondary circuit. These amoebots become passive amoebots in the next iteration. The algorithm terminates when the second round is silent. At this point, amoebot u_r is the remaining active amoebot. Feldmann et al. (2022) have proven the following lemma.

Lemma 3.1 *The PASC algorithm terminates in* $\lceil \log m \rceil$ *iterations, resp.* $O(\log m)$ *rounds.*

We obtain the identifiers from the PASC algorithm as follows. Let $k = \lceil \log m \rceil$ be the number of iterations. For $0 \le i < k$, let r_i be the first round of the $(i+1)$-st iteration. Note that in each of these rounds, each amoebot of the chain receives a beep either on its primary circuit or its secondary circuit. Hence, in round r_i, amoebot x interprets a beep on the primary circuit as $x_i = 0$ and a beep on the secondary circuit as $x_i = 1$.

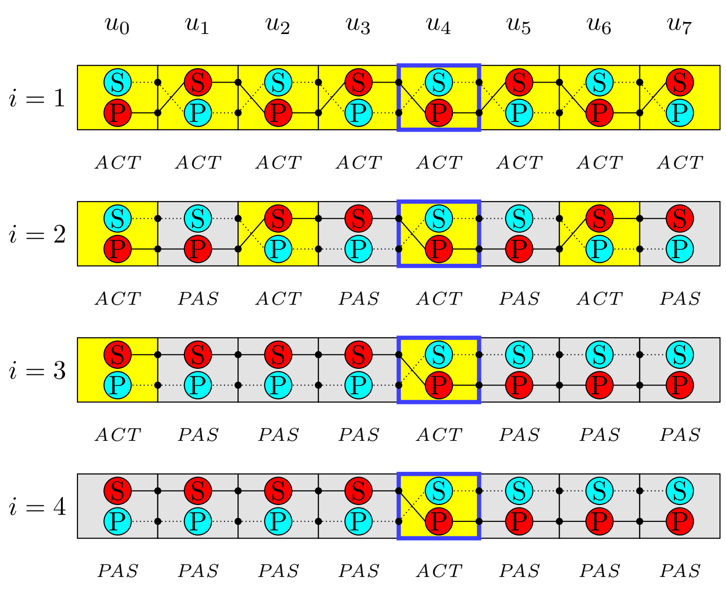
Lemma 3.2 *Given a chain C of amoebots and an amoebot u_r of the chain, the PASC algorithm computes* $\mathrm{id}_{C,u_r}(v)$ *for each amoebot v in the chain.*

Proof Since the reference amoebot u_r activates its primary circuit, it always receives a beep on its primary circuit. This implies $(u_r)_i = 0$ for $0 \le i < k$. Thus, Eq. 1 holds.

Let y be the successor of x. We have to show that Eq. 2 holds, i.e., $\mathrm{id}_{C,u_r}(y) = \mathrm{id}_{C,u_r}(x) + 1$.

We have two cases: (i) y never becomes passive, and (ii) y becomes passive. The first case directly implies

[2] In comparison to Feldmann et al. (2022), we have adjusted the rules by mirroring the connections.



u_0 u_1 u_2 u_3 u_4 u_5 u_6 u_7
$i = 1$
ACT ACT ACT ACT ACT ACT ACT ACT
$i = 2$
ACT PAS ACT PAS ACT PAS ACT PAS
$i = 3$
ACT PAS PAS PAS ACT PAS PAS PAS
$i = 4$
PAS PAS PAS PAS ACT PAS PAS PAS
S
P

Fig. 5 PASC algorithm. Each figure shows the chain at the beginning of the i-th iteration. The circles indicate the primary (P) and secondary (S) partition sets. The blue bordered amoebot denotes the reference amoebot u_r. Yellow amoebots are active, and gray amoebots are passive. The solid edges indicate the primary circuit of u_r, and the dotted edges the secondary circuit of u_r. All primary and secondary partition sets that are part of the primary resp. secondary circuit of u_r are depicted in red resp. cyan

$\operatorname{id}_{C,u_r}(x) = (1,\ldots,1) = -1$ and $\operatorname{id}_{C,u_r}(y) = (0,\ldots,0) = 0$. For the second case, let l denote the iteration where y becomes passive. Hence, y has received a beep on its primary circuit in the first $l-2$ iterations and a beep on its secondary circuit in the $(l-1)$-st iteration, i.e., $y_{l-1} = 1$ and $y_i = 0$ for $0 \le i < l-1$. Since y is active, x has received a beep on its secondary circuit in the first $l-2$ iterations and a beep on its primary circuit in the $(l-1)$-st iteration, i.e., $x_{l-1} = 0$ and $x_i = 1$ for $0 \le i < l-1$. Since y is passive from the l-th iteration, $x_i = y_i$ holds for $l \le i < k$. We obtain

$$\begin{aligned}\operatorname{id}_{C,u_r}(x) &= (x_{k-1},\ldots,x_l,0,1,\ldots,1)\\ \operatorname{id}_{C,u_r}(y) &= (y_{k-1},\ldots,y_l,1,0,\ldots,0)\\ &= (x_{k-1},\ldots,x_l,0,1,\ldots,1)+1\end{aligned}$$

since $2^{l-1} = \sum_{i=0}^{l-2} 2^i + 1$. □

3.2 Spatial identifiers

In this section, we compute identifiers relative to the spatial positions of the amoebots with respect to a cardinal direction $d \in D_m \cup D_p$ and a reference amoebot $u_r \in S$. Let d' denote the direction obtained if we rotate d by 90° counterclockwise, e.g., $d' = N$ for $d = E$. First, consider $d \in D_p$ (see left side of Fig. 6). Let $\mathcal{A}_d = \{A(S,v,d') \mid v \in S\}$ denote a set of stripes. We first assign identifiers to $\mathcal{A}_d$. Afterwards, we extend these identifiers to the nodes in S.

Observe that $\mathcal{A}_d$ partitions S into disjoint stripes. These stripes form a chain $\mathcal{C}_d$ if we think of the stripes as nodes such that two nodes are adjacent if the corresponding stripes are neighbors (see left side of Fig. 6). The order of the chain is given by the cardinal direction d: The successor of a stripe is the neighboring stripe in direction d. Let $\operatorname{succ}(A)$ denote the succeeding stripe of stripe A. Let $A_r = A(S,u_r,d')$. We assign identifiers $\operatorname{id}_{\mathcal{C}_d,A_r}$ according to the following two rules:

$$\begin{aligned}\operatorname{id}_{\mathcal{C}_d,A_r}(A_r) &= 0\\ \operatorname{id}_{\mathcal{C}_d,A_r}(\operatorname{succ}(A)) &= \operatorname{id}_{\mathcal{C}_d,A_r}(A)+1 \text{ for all } A \in \mathcal{A}_d\end{aligned}$$

Finally, we define $\operatorname{id}_{d,u_r}(v) = \operatorname{id}_{\mathcal{C}_d,A_r}(A(S,v,d'))$ for all nodes $v \in S$.

In order to compute the identifiers, we transfer the concept of primary and secondary circuits from a chain of amoebots to a chain of stripes (compare with Sect. 3.1): From a global perspective, each stripe knows its predecessor and its successor, is either active or passive, and has a primary and secondary partition set. The primary and secondary partition sets are connected by the following rules: If a stripe is active, its primary partition set is connected to the secondary partition set of its predecessor, and its secondary partition set is connected to the primary partition set of its predecessor. If a stripe is passive, its primary partition set is connected to the primary partition set of its predecessor, and its secondary partition set is connected to the secondary partition set of its predecessor. There are no further connections. In order to keep the amoebots within a stripe synchronized, we additionally require that each amoebot has access to the primary and secondary partition set of its stripe. In the following, we explain how to construct the circuits that satisfy the aforementioned properties.

Note that the neighborhood of each amoebot only contains amoebots of the same stripe, the preceding stripe, and the succeeding stripe. Since we assume common compass orientation and chirality, each amoebot is able to determine to which stripe each neighbor belongs. We now define pin configurations that satisfy the aforementioned properties.

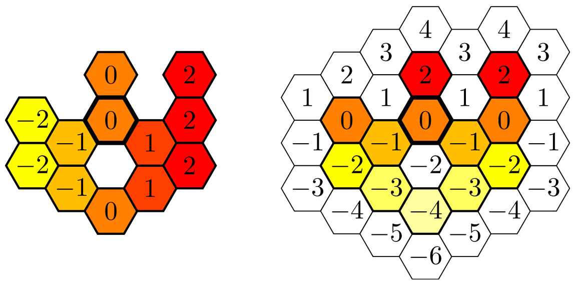


Fig. 6 Spatial identifiers. The left figure show the spatial identifiers with respect to $d = E$, and right figure with respect to $d = N$. Each color indicates a stripe (in S). The white hexagons indicate the neighborhood of S. The thick boundary indicates the reference amoebot u_r

Each pin configuration has two partition sets that we call the primary and secondary partition set. We connect the primary and secondary partition sets of an amoebot to the primary and secondary partition sets of adjacent amoebots such that the aforementioned properties are reflected locally. For example, consider two adjacent amoebots u and v such that u belongs to an active stripe and v belongs to u's preceding stripe. We connect u's primary partition set to v's secondary partition set, and u's secondary partition set to v's primary partition set. Since we assign the same identifiers to amoebots of the same stripe, we connect their primary and secondary partition sets, respectively. We obtain two pin configurations: one for amoebots of active stripes and one for amoebots of passive stripes (see Fig. 7).

Lemma 3.3 *The construction satisfies the aforementioned properties.*

Proof For the sake of analysis, we first consider an infinite amoebot structure where $S = V$. Afterwards, we transfer the results to arbitrary connected amoebot structures.

Consider a single stripe. Let *ACTIVE* denote the pin configuration used for amoebots of active stripes, and *PASSIVE* denote the pin configuration used for amoebots of passive stripes. Both pin configurations define a primary and secondary partition set. All amoebots within the stripe connect their primary and secondary partition sets, respectively (see Fig. 8). We define the union of all primary resp. secondary partition sets within a stripe as the primary resp. secondary partition set of the stripe. Note that each amoebot has access to both partition sets and is able to distinguish between them.

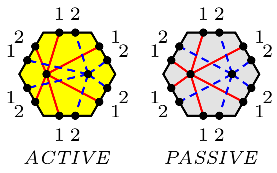


Fig. 7 Utilized pin configurations for $d = E$

Next, consider the connections to the preceding stripe. The connections within the pin configuration *ACTIVE* are selected in such a way that an active stripe connects its primary partition set exclusively to the secondary partition set of the preceding stripe, and its secondary partition set exclusively to the primary partition set of the preceding stripe (see Fig. 8). Similar, the connections within the pin configuration *PASSIVE* are selected in such a way that an active stripe connects its primary partition set exclusively to the primary partition set of the preceding stripe, and its secondary partition set exclusively to the secondary partition set of the preceding stripe (see Fig. 8). Note that there are no further connections.

The crucial property of our construction is that any two amoebots are connected by any arbitrary path of amoebots in the same fashion, i.e., either both primary partition sets are connected to the secondary partition set of the other amoebot, or their primary and secondary partition sets are connected, respectively (see left side of Fig. 9). This allows us to remove amoebots without separating the circuits as long as the amoebot structure stays connected (see right side of Fig. 9). □

Now, we simply apply the PASC algorithm on the chain of stripes to compute $\mathrm{id}_{\mathcal{C}_d, A_r}$, i.e., each amoebot $v \in S$ computes $\mathrm{id}_{\mathcal{C}_d, A_r}(\mathrm{A}(S, v, d'))$ that equals $\mathrm{id}_{d,u_r}(v)$ by definition.

Lemma 3.4 *Given $u_r \in S$ and $d \in D_p$, the PASC algorithm computes $\mathrm{id}_{d,u_r}(v)$ for each $v \in S$.*

Proof By Lemma 3.3, we can apply the primitive of primary and secondary circuits to the chain of stripes as long as each amoebot knows whether its stripe is active or passive. This implies that we can perform the PASC algorithm that by Lemma 3.2, computes $\mathrm{id}_{\mathcal{C}_d, A_r}$.

It remains to show that each amoebot knows whether its stripe is active or passive throughout the execution of the algorithm. Initially, this is trivially true since each stripe is active. A stripe becomes passive once it receives a beep on its secondary circuit. Each amoebot of the stripe can observe this beep since by construction, each amoebot has access to the secondary partition set of its stripe. Thereafter, the stripe stays passive. □

Next, consider $d \in D_m$. We discuss the necessary modifications in comparison to $d \in D_p$. Note that an amoebot is unable to locally determine to which stripe its neighbors belong (see Fig. 10). We therefore perform the PASC algorithm on the union of the amoebot structure and its neighborhood (see right side of Fig. 6). Note that each neighbor of an amoebot $v \in S$ belongs either to one of the

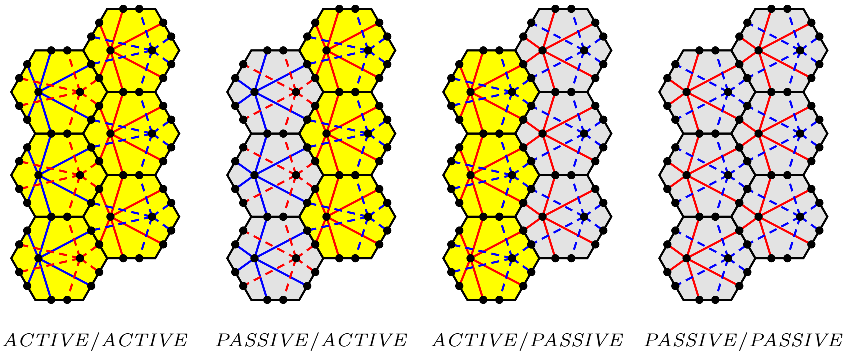


Fig. 8 Connectivity between adjacent stripes for $d = E$. The figures show all possible combinations of active and passive stripes

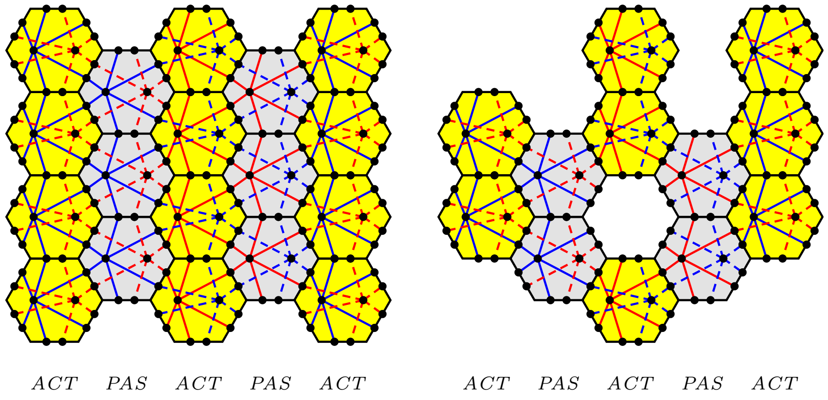


Fig. 9 Construction for $d = E$. The left figure shows a section of the infinite amoebot structure, and the right figure an arbitrary amoebot structure. The connectivity of the remaining amoebots is preserved

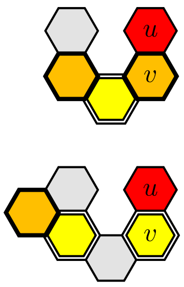


Fig. 10 Neighboring stripes. Amoebot u is unable to determine whether amoebot v belongs to the preceding stripe (orange, thick boundary) or the stripe preceding its preceding stripe (yellow, double boundary)

two preceding stripes or to one of the two succeeding stripes.

Each amoebot tracks when the stripes of its neighbors become passive as follows. Recall that all stripes are initially active. Suppose that each amoebot $v \in S$ knows whether (its stripe and) the stripes of its neighbors are active or passive at the beginning of an iteration of the PASC algorithm. Hence, v also knows how these stripes are interconnected. Thus, v can conclude from the signal it receives on the signals received by the stripes of its neighbors and with that whether these become passive.

There are two reasons for the tracking. First, some of the stripes are not occupied by any amoebots. The tracking allows each amoebot $v \in S$ to activate the correct circuits for each of its neighbors. Second, the connections between an amoebot and its neighbor of the stripe preceding its preceding stripe depends on the states of its stripe and its preceding stripe. We obtain four pin configurations (see Fig. 11).

Lemma 3.5 *The construction satisfies the aforementioned properties.*

Proof The proof is similar to the one of Lemma 3.3. For the sake of analysis, we first consider an infinite amoebot structure where $S = V$. Afterwards, we transfer the results to arbitrary connected amoebot structures.

Consider a single stripe A_1. All pin configurations define a primary and secondary partition set. Note that the amoebots within the stripe are not adjacent. So, we cannot connect them directly. However, we still define the union of all primary resp. secondary partition sets within a stripe as the primary resp. secondary partition set of the stripe. Note that each amoebot has access to both partition sets and is able to distinguish between them.

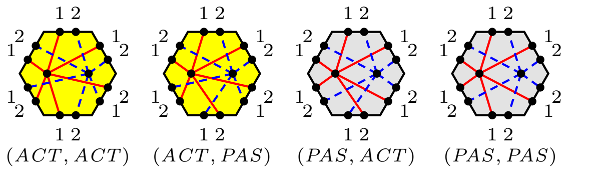


Fig. 11 Utilized pin configurations for $d = N$. The first argument denotes the state of the amoebot's stripe and the second argument denotes the state of the preceding stripe, respectively

Next, consider the connections to its preceding stripe A_2 and the stripe A_3 preceding A_2. If A_1 and A_2 are active, then A_1 utilize pin configuration (ACT, ACT). The connections are selected in such a way that A_1 connects its primary partition set exclusively to the secondary partition set of A_2, and its secondary partition set exclusively to the primary partition set of A_2 (see Fig. 12). Further, A_1 connects its primary partition set exclusively to the primary partition set of A_3, and its secondary partition set exclusively to the secondary partition set of A_3 (see Fig. 12).

If A_1 is active and A_2 is passive, then A_1 utilize pin configuration (ACT, PAS). The connections are selected in such a way that A_1 connects its primary partition set exclusively to the secondary partition set of A_2, and its secondary partition set exclusively to the primary partition set of A_2 (see Fig. 12). Further, A_1 connects its primary partition set exclusively to the secondary partition set of A_2, and its secondary partition set exclusively to the primary partition set of A_2 (see Fig. 12).

If A_1 is passive and A_2 is active, then A_1 utilize pin configuration (PAS, ACT). The connections are selected in such a way that A_1 connects its primary partition set exclusively to the primary partition set of A_2, and its secondary partition set exclusively to the secondary partition set of A_2 (see Fig. 12). Further, A_1 connects its primary partition set exclusively to the secondary partition set of A_3, and its secondary partition set exclusively to the primary partition set of A_3 (see Fig. 12).

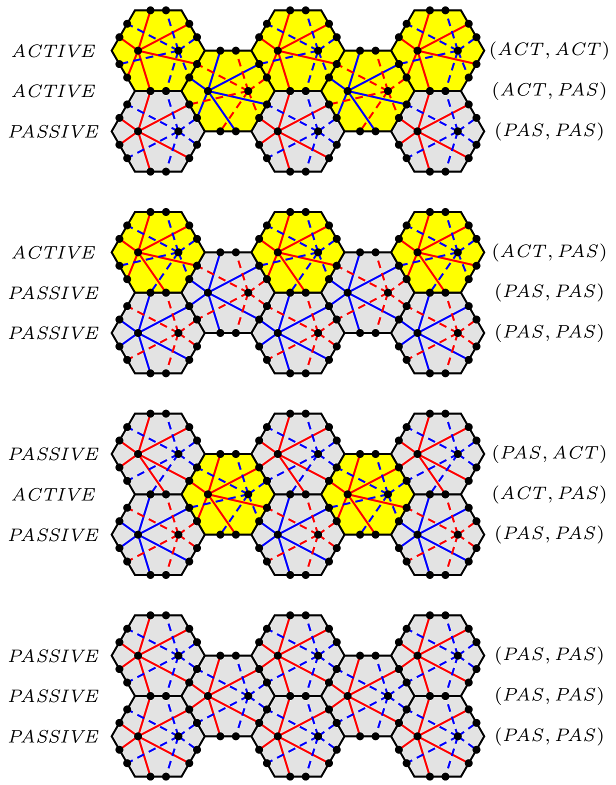


Fig. 12 Connectivity between adjacent stripes for $d = N$. The left label indicates whether the stripe is active or passive, and the right label indicates the utilized pin configuration. We only depict the combinations of three consecutive stripes where the southernmost stripe and its preceding stripe are passive, i.e., the southernmost stripe utilizes pin configuration (PAS, PAS)

If A_1 and A_2 are passive, then A_1 utilize pin configuration (PAS, PAS). The connections are selected in such a way that A_1 connects its primary partition set exclusively to the primary partition set of A_2, and its secondary partition set exclusively to the secondary partition set of A_2 (see Fig. 12). Further, A_1 connects its primary partition set exclusively to the primary partition set of A_3, and its secondary partition set exclusively to the secondary partition set of A_3 (see Fig. 12).

Hence, the aforementioned properties hold between each stripe and its preceding stripe and between each stripe and the stripe preceding its preceding stripe. This ensures that each amoebot connects its primary and secondary partition set to the primary and secondary partition set of each of its neighbors. Also note that our construction has connected all primary and secondary partition sets within each stripe, respectively (see Fig. 12).

The crucial property of our construction is that any two amoebots are connected by any arbitrary path of amoebots in the same fashion, i.e., either both primary partition sets are connected to the secondary partition set of the other amoebot, or their primary and secondary partition sets are connected, respectively (see left side of Fig. 13). This allows us to remove amoebots without separating the circuits as long as the amoebot structure stays connected (see right side of Fig. 13). □

Lemma 3.6 *Given $u_r \in S$ and $d \in D_m$, the PASC algorithm computes $\text{id}_{d,u_r}(v)$ for each $v \in S$.*

Proof The proof works analogously to the one of Lemma 3.4. By Lemma 3.5, we can apply the primitive of primary and secondary circuits to the chain of stripes as long as each amoebot knows whether its stripe and the stripe of each of its neighbors are active or passive. Note that the knowledge about the latter allows each amoebot to simulate non-occupied nodes in its neighborhood, which prevents gaps in the chain $\mathcal{C}_d$. This implies that we can perform the PASC algorithm that by Lemma 3.2, computes $\text{id}_{\mathcal{C}_d, A_r}$.

It remains to show that each amoebot knows whether its stripe and the stripe of each of its neighbors are active or passive throughout the execution of the algorithm. Initially, this is trivially true since each stripe is active. A stripe becomes passive once it receives a beep on its secondary circuit. Each amoebot of the stripe can observe this beep since by construction, each amoebot has access to the secondary partition set of its stripe. Thereafter, the stripe stays passive.

Further, each amoebot knows how its stripe and the stripe of each of its neighbors are connected. Thus, it can conclude from the signal it receives on the signals received by the stripe of its neighbors and with that when that stripe becomes passive. Thereafter, the stripe stays passive. □

Note that we can generalize this technique to arbitrary directions as long as the identifiers within a neighborhood only differ by some constant.

3.3 Applications

We now consider two applications for the identifiers, namely the stripe problem and the global maxima problem.

First, consider the stripe problem. Recall that we compute the set of amoebots on the axis through u in direction d. Let d' denote the direction obtained if we rotate d by 90° clockwise, e.g., $d' = E$ for $d = N$. Our stripe algorithm simply executes the PASC algorithm with respect to direction d' and with u as the reference amoebot, i.e., $u_r = u$. Note that this sets the identifier of u to 0, i.e., $\mathrm{id}_{d',u}(u) = 0$. By construction, $\mathrm{id}_{d',u}(v) = \mathrm{id}_{d',u}(u) = 0$ holds for all $v \in \mathrm{A}(S, u, d)$. We obtain the following theorem.

Theorem 3.7 *The stripe algorithm solves the stripe computation problem in $O(\log n)$ rounds.*

Remark 3.8 By construction, each amoebot in direction d' from $\mathrm{A}(S, u, d)$ receives a positive identifier, and each amoebot in the opposite direction from $\mathrm{A}(S, u, d)$ receives a negative identifier (e.g., see Fig. 6). The sign of an identifier is given by the most significant bit of its two's complement representation. More precisely, the identifier is positive if the most significant bit is 0 and at least one other bit is 1, and negative if the most significant bit is 1. Hence, each amoebot $v \notin \mathrm{A}(S, u, d)$ is able to determine on which side of the axis $\mathrm{X}(u, d)$ it is situated.

Next, consider the global maxima problem. Recall that we compute the global maxima of a set $R \subseteq S$ (see Sect. 1.3) with respect to direction d. By construction,

$$\operatorname{argmin}_{w \in R} \mathrm{f}_d(R, w) = \operatorname{argmax}_{w \in R} \mathrm{id}_{d,u_r}(w)$$

holds for any reference amoebot u_r. The idea of our global maxima algorithm is therefore to execute the PASC algorithm and to determine the highest identifier.

We first elect an $u \in R$ by applying a leader election (see Theorem 2.1). By choosing u as the reference amoebot, i.e., $u_r = u$, we ensure that the maximal identifier is non-negative. In order to determine the maximum of non-negative numbers, we apply the consensus algorithm by Feldmann et al. (2022) that agrees on the highest input value. First, the amoebot structure establishes the global circuit (see Sect. 2). Each amoebot $v \in R$ with $\mathrm{id}(v) \geq 0$ transmits its identifier starting from the most significant bit. If a transmitting amoebot observes a beep in a round it does not beep, it stops its transmission. Only an amoebot with the highest identifier is able to transmit its identifier until the end.

However, since each amoebot can only store a constant section of its identifier, and since the PASC algorithm provides the identifiers from the least significant bit to the most significant bit, we have to partially recompute the identifiers after each bit. In order to identify the correct bit, we simply use two binary counters where the first counter indicates the current bit, and the second counter the current iteration of the PASC algorithm. We must be able to increment, decrement, and compare the counters.

We realize the counters along a chain of amoebots as follows. The i-th amoebot in the chain holds the i-th bit of both counters, respectively. Initially, each bit is set to 0. In order to increment (decrement) a counter, we have to flip the first 0 (1) and all preceding bits. For that, we construct a circuit along the chain that we cut at each amoebot holding a 1 (0). Then, the first amoebot in the chain beeps. We interpret the beep as a carryover. Hence, each amoebot receiving the beep flips its bit. In order to compare the counters, we utilize the global circuit. Each amoebot of the chain holding two different bits beeps on the circuit. The

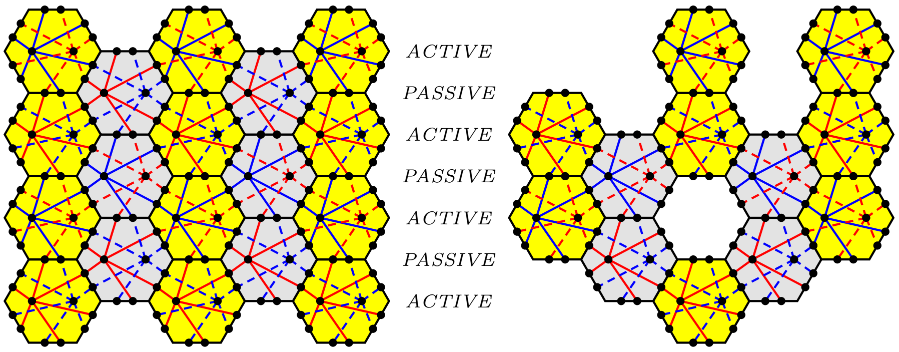


Fig. 13 Construction for $d = N$. The left figure shows a section of the infinite amoebot structure, and the right figure an arbitrary amoebot structure. The connectivity of the remaining amoebots is preserved

counters are equal iff no amoebot beeps. All three operations require $O(1)$ rounds. Using a chain along the outer boundary set ensures a sufficient length of the counters. We obtain the following theorem.

Theorem 3.9 *The global maxima algorithm computes the global maxima within $O(\log^2 n)$ rounds w.h.p.*

Proof By Theorem 2.1, the leader election requires $O(\log n)$ rounds w.h.p. Then, we proceed iteratively. In each iteration, we apply the PASC algorithm and apply a single round of the consensus algorithm. By Lemmas 3.4 and 3.6, the PASC algorithm computes the identifiers within $O(\log n)$ rounds. The consensus algorithm by Feldmann et al. (2022) only adds a single round to each iteration. Since we need $O(\log n)$ iterations (one for each bit), the global maxima algorithm requires $O(\log^2 n)$ rounds w.h.p. □

3.4 Distances in the triangular grid graph

In this section, we compute identifiers equal to the distances in G_Δ between the amoebots and a reference amoebot $u_r \in S$. Let $\operatorname{dist}_\Delta(u, v)$ denote the distance between u and v in G_Δ. The idea is to once again apply the PASC algorithm. In the following, we will focus on the necessary modifications in comparison to Sect. 3.2. Instead of a chain of stripes, we utilize a chain of sets with amoebots of the same distance to the reference amoebot (see Fig. 14). More precisely, let the k-th set of the chain be the set of amoebots with distance $k-1$ to the reference amoebot u_r. In particular, the first set only contains u_r. Note that a set is not necessarily connected.

In order to correctly interconnect the primary and secondary partition sets, each amoebot $v \in S$ has to determine which of its neighbors belong to the same set, the preceding set, and the succeeding set. For that, consider the rays from u_r into each direction $d \in D_m$. These divide G_Δ into six triangles. The classification of v's neighborhood depends on which ray or triangle it is on.

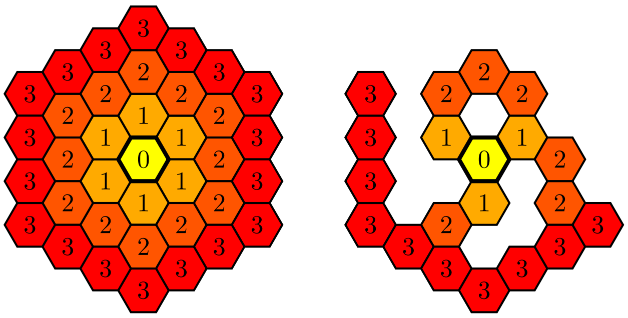


Fig. 14 Distances in G_Δ. Each color indicates a set of the chain. The thick boundary indicates the reference amoebot u_r

For example, consider an amoebot on the ray into direction N (see Fig. 14). The amoebots into directions WNW, N, and ENE belong to the succeeding set. The amoebots into directions WSW, and ESE belong to the same set. The amoebot into direction S belongs to the preceding set.

For another example, consider an amoebot on a triangle between the rays into directions ENE and ESE (see Fig. 14). The amoebots into directions ENE, and ESE belong to the succeeding set. The amoebots into directions N, and S belong to the same set. The amoebots into directions WNW, and WSW belong to the preceding set. Note that this corresponds to the neighborhood for spatial identifiers with respect to direction E.

In order to determine the ray or triangle each amoebot is on, we can simply perform the stripe algorithm with u_r as the reference amoebot and for each $d \in \{N, ESE, WSW\}$. By Remark 3.8, the received identifiers indicate the position of each amoebot (see Fig. 15).

We interconnect the primary and secondary partition sets by the same rules as for spatial identifiers: If the set is active, its primary partition set is connected to the secondary partition set of its predecessor, and its secondary partition set is connected to the primary partition set of its predecessor. If the set is passive, its primary partition set is connected to the primary partition set of its predecessor, and its secondary partition set is connected to the secondary partition set of its predecessor. In total, we receive 13 different pin configurations (see Fig. 16). We obtain the following lemmas.

Lemma 3.10 *The construction satisfies the properties described in Sect. 3.2.*

Proof The proof works analogously to the one of Lemma 3.3. For the sake of analysis, we first consider an infinite amoebot structure where $S = V$. Afterwards, we transfer the results to arbitrary connected amoebot structures.

Consider a single set. All pin configurations define a primary and secondary partition set. All amoebots within the set connect their primary and secondary partition sets, respectively (see Fig. 17). We define the union of all primary resp. secondary partition sets within a set as the primary resp. secondary partition set of the set. Note that each amoebot has access to both partition sets and is able to distinguish between them.

Next, consider the connections to the preceding set. The connections within the pin configurations for active amoebots are selected in such a way that an active set connects its primary partition set exclusively to the secondary partition set of the preceding set, and its secondary partition set exclusively to the primary partition set of the preceding set (see Fig. 17). Similar, the

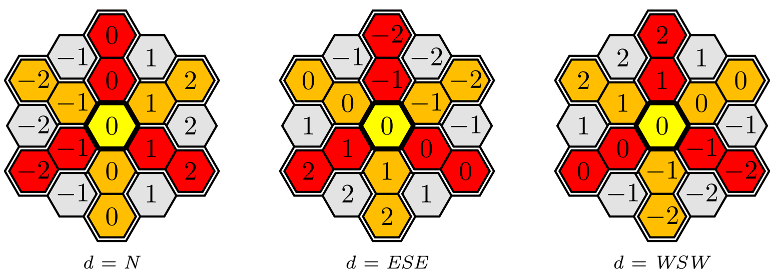


Fig. 15 Rays and triangles. The thick boundary indicates the reference amoebot u_r. The red and orange amoebots (double boundary) indicate the six rays. The gray amoebots indicate the triangles. Note that only amoebots on the rays receive 0 as an identifier. For example, an amoebot on the ray into direction N receives 0 as identifier for $d = N$, a negative identifier for $d = ESE$, and a positive identifier for $d = WSW$. For another example, an amoebot on a triangle between the rays into directions ENE and ESE receives a positive identifier for $d = N$, and negative identifiers for $d = ESE$ and $d = WSW$, respectively

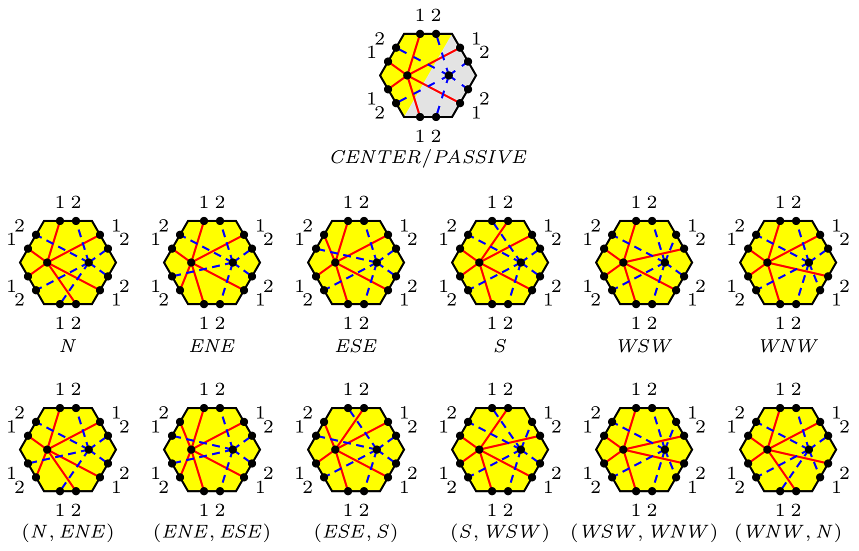


Fig. 16 Utilized pin configurations for the computation of distances in G_Δ. The first pin configuration is used for the reference amoebot in the center and passive amoebots. The second row shows the pin configurations for active amoebots on the rays. The direction below each amoebot indicates the direction of the ray it is on. The third row shows the pin configurations for active amoebots on the triangles. The directions below each amoebot indicate the directions of the incident rays

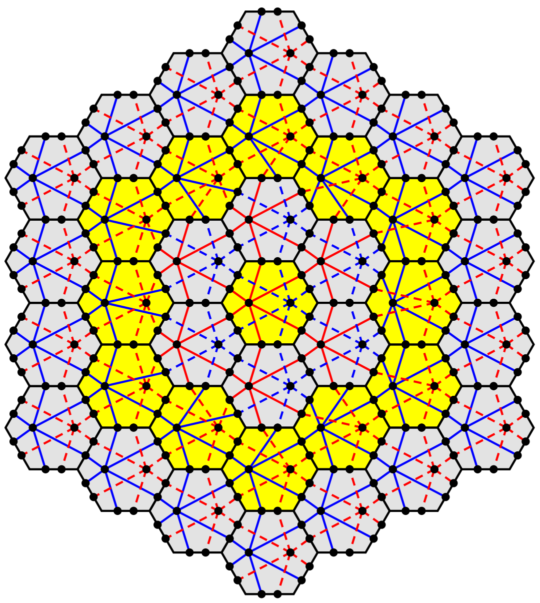
connections within the pin configuration for passive amoebots are selected in such a way that an active set connects its primary partition set exclusively to the primary partition set of the preceding set, and its secondary partition set exclusively to the secondary partition set of the preceding set (see Fig. 17). Note that there are no further connections.

The crucial property of our construction is that any two amoebots are connected by any arbitrary path of amoebots in the same fashion, i.e., either both primary partition sets are connected to the secondary partition set of the other amoebot, or their primary and secondary partition sets are connected, respectively (see top of Fig. 17). This allows us to remove amoebots without separating the circuits as long as the amoebot structure stays connected (see bottom of Fig. 17). □

Lemma 3.11 *Given $u_r \in S$, the PASC algorithm computes* $\operatorname{dist}_\Delta(u_r, v)$ *for each $v \in S$.*

Proof The proof works analogously to the one of Lemma 3.4. By Lemma 3.10, we can apply the primitive of primary and secondary circuits to the chain of sets as long as each amoebot knows whether its set is active or passive. This implies that we can perform the PASC algorithm that by Lemma 3.2, computes the distance of each set to u_r.

It remains to show that each amoebot knows whether its set is active or passive throughout the execution of the algorithm. Initially, this is trivially true since each set is active. A set becomes passive once it receives a beep on its secondary circuit. Each amoebot of the set can observe this beep since by construction, each amoebot has access to the secondary partition set of its set. Thereafter, the stripe stays passive. □



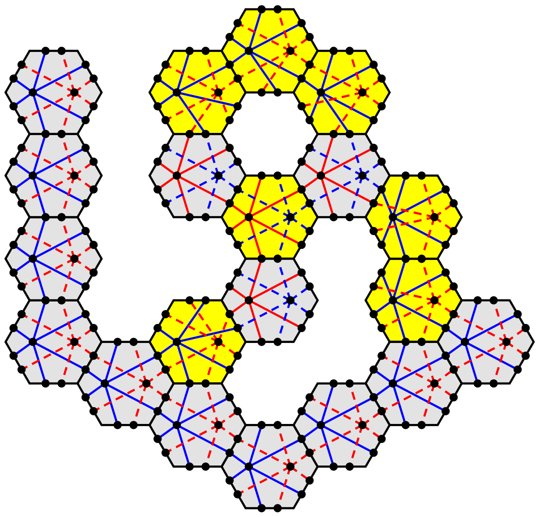


Fig. 17 Construction for distances in G_Δ. The top figure shows a section of the infinite amoebot structure, and the bottom figure an arbitrary amoebot structure. The connectivity of the remaining amoebots is preserved. The reference amoebot in the center and all amoebots with distance 2 to the reference amoebot are active while all other amoebots are passive. The top figure includes all 13 pin configurations. Note that the pin configuration of all passive amoebots and of the reference amoebot in the center are identical

4 Skeletons

This section deals with (canonical) skeletons. In the first subsection, we canonicalize and construct the skeletons. In the second and third subsection, we show two applications for skeletons by showing how to construct spanning trees and how to detect symmetries.

4.1 Canonicalized construction

The general idea to construct a skeleton is to start with the cycles given by the boundary sets and to fuse these into a single cycle, i.e., a skeleton (see Fig. 18). In order to fuse two boundary cycles, we have to determine a path between them. The boundary cycles are split at the respective endpoints of the path and connected along the path. Note that the fused cycle uses the path twice. The difficulty lies in finding paths between the boundary cycles and in avoiding the creation of new cycles. The construction is correct, i.e., we obtain a single cycle, iff the boundary sets and paths form a tree with the boundary sets as the nodes and the paths as the edges. In order to obtain a skeleton path, the skeleton is split at an arbitrary point, e.g., chosen by a leader election.

We now canonicalize the construction of a skeleton and a skeleton path. Recall that we define the canonical skeleton (path) with respect to a cardinal direction $d \in D_m \cup D_p$ and a sign $s \in \{+,-\}$. For the canonical skeleton, we have to define how the paths are constructed, and how the boundary cycles and paths are exactly connected. Given a skeleton, we obtain a skeleton path by splitting the skeleton at some point. For the skeleton path to be canonical, we have to identify a canonical point for the splitting. Afterwards, we show how the amoebot structure computes the canonical skeleton in a distributed fashion.

We start with the construction of the canonical skeleton. Let $\rho_s(d,x)$ denote the direction obtained if we rotate direction d by x degrees counterclockwise if the sign s is positive, and clockwise if the sign s is negative. Let $d_p = d$ if $d \in D_m$ and $d_p = \rho_s(d,30)$ if $d \in D_p$. For each inner boundary set B, we construct a path as follows. First, we compute the global maxima of B with respect to direction d. Let B_d denote these global maxima. Then, we compute the global maximum of B_d with respect to direction $\rho_s(d,90)$. Let u_B denote the global maximum. Let R denote the empty region enclosed by B (see Sect. 2).

Lemma 4.1 *Amoebot u_B is adjacent to exactly one node in R, namely the one in direction $\rho_s(d_p,180)$. Further, no amoebot in direction d_p of u_B is adjacent to a node in R.*

Proof Amoebot u_B has to be adjacent to at least one node in R. Otherwise, $u_B \notin B$ would hold. It is easy to see that if that node would lie in another direction than $\rho_s(d_p,180)$, either u_B would not be a global maximum of B with respect to d, or u_B would not be the global maximum of B_d with respect to direction $\rho_s(d,90)$. By the same reasoning, u_B would not be a global maximum of B with respect to d if any amoebot in direction d_p of u_B would be adjacent to a node in R. □

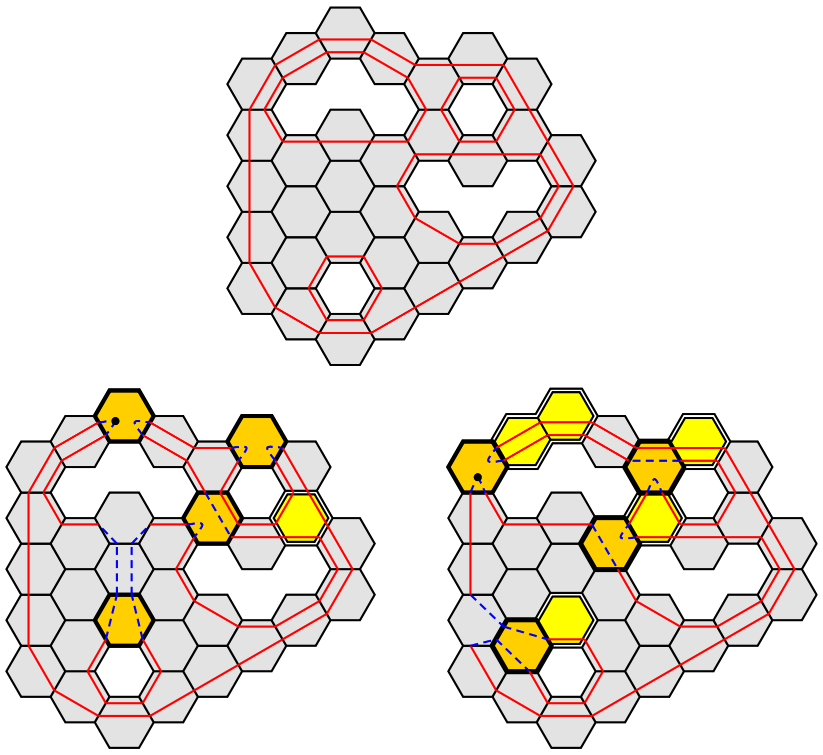


Fig. 18 Canonical skeleton. The top figure shows the initial situation. The red lines indicate the boundary cycles. The bottom figures show the $(N, +)$-skeleton and $(NNW, +)$-skeleton, respectively. The yellow and orange amoebots (thick or double boundary) indicate the global maxima of the boundary sets. The orange amoebots (thick boundary) indicate the starting points of the paths. The blue dashed lines indicate the paths between the boundary cycles. The node indicates the location where the cycle is split

The path starts at u_B and goes straight in direction d_p until it reaches an amoebot v_B of another boundary set. The existence of v_B is guaranteed by the outer boundary set. Clearly, all nodes of the path are occupied by amoebots. Note that the path may be trivial, i.e., $u_B = v_B$. There is only a single case where v_B is part of two boundary sets unequal to B. In this case, we take the boundary of v_B in direction $\rho_s(d_p, 60)$.

Lemma 4.2 *The boundary sets and paths form a tree.*

Proof Let $\text{rank}(B) = \min_{w \in B} \text{f}_d(S, w)$ be the rank of an inner boundary set B, and let $\text{rank}(B_O) = -1$ be the rank of the outer boundary set B_O (compare to the definition of the global maxima in Sect. 1.3). The rank of an inner boundary set is lower than the rank of another inner boundary set if its global maxima are further in direction d than the global maxima of the other inner boundary set. Furthermore, the outer boundary set has a lower rank than all ranks of the inner boundary sets.

We claim that for each inner boundary set B, we construct a path from B to another boundary set B' such that $\text{rank}(B) > \text{rank}(B')$ holds. Clearly, this relationship cannot be cyclic. The lemma immediately follows since we construct a path for each inner boundary set. We prove the claim in the following.

Lemma 4.1 excludes the possibility of a self-loop, i.e., $B \neq B'$ holds. The claim holds by definition if B' is the outer boundary set. Suppose that B' is an inner boundary set. The claim also holds if the path from u_B to v_B is not trivial since $\text{rank}(B) = \text{f}_d(S, u_B) > \text{f}_d(S, v_B) > \text{rank}(B')$ holds.

Suppose that the path is trivial, i.e., $u_B \in B$ and $u_B \in B'$. Let $w \in R'$ be a node adjacent to u_B. Note that $\text{f}_d(S, u_B) \geq \text{f}_d(S, w)$ holds since otherwise, $R = R'$ and with that $B = B'$ would hold. We go from w into direction d_p until we reach an amoebot $x \in B'$. Note that $\text{f}_d(S, w) > \text{f}_d(S, x)$ holds since for $V \setminus R_O$, f_d is strictly monotonically decreasing if we go into direction d_p. This amoebot exists since B' is an inner boundary set. The claim holds since $\text{rank}(B) = \text{f}_d(S, u_B) \geq \text{f}_d(S, w) > \text{f}_d(S, x) > \text{rank}(B')$ holds. □

It remains to define how the cycles and paths are exactly connected. We define that the cycle runs along the tree without crossing itself.

Next, consider the construction of the canonical skeleton path. We determine the splitting point u_{B_O} by applying the same procedure as for the starting points of the paths on the outer boundary set B_O. That is, we first compute the global maxima of B_O with respect to direction d, and then compute the global maximum of these global maxima with respect to direction $\rho_s(d, 90)$. If the canonical skeleton visits u_{B_O} multiple times, we pick a predefined position with respect to d_p (see Fig. 19).

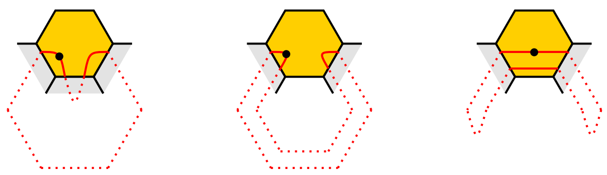


Fig. 19 Predefined splitting point with respect to d_p. For $d_p = N$, the figure shows all cases where the canonical skeleton visits u_{B_O} multiple times. By similar arguments as for Lemma 4.1, there are no other cases. The red lines indicate the canonical skeleton. The node indicates the splitting point

Lemma 4.3 *The canonical skeleton (path) visits each bond at most twice. Thus, the canonical skeleton has linear complexity.*

Proof The canonical skeleton (path) visits a bond either due to a boundary cycle or due to one of the paths. Each local boundary (a common unoccupied adjacent node of the endpoints) adds one visit. Each bond has at most two local boundaries. Each path adds two visits. Due to Lemma 4.1, a bond cannot be part of more than one path. A bond cannot be part of a boundary cycle and a path at the same time since the path would stop at either endpoints due to the unoccupied adjacent node. Altogether, each bond is visited at most twice. □

In the following, we present our canonical skeleton algorithm that computes the canonical skeleton and the splitting point for the canonical skeleton path in parallel. Some instructions are performed on different subsets in parallel. In order to keep the amoebot structure synchronized, we apply the synchronization primitive (see Sect. 2). In a preprocessing step, each boundary set determines whether it is an inner or outer boundary set (see Corollary 2.3).

The canonical skeleton algorithm follows our construction of the canonical skeleton. In the first step, we compute the starting points of the paths and the splitting point by performing the global maxima algorithm on each boundary set B with respect to direction d, and on each resulting set B_d with respect to direction $\rho_s(d, 90)$. However, the computation of the boundary sets may interfere with each other since the boundary sets may intersect. In order to circumvent that problem, we add two additional external links and let each boundary set use the two external links closer to the corresponding empty region (see Fig. 20). Note that an amoebot is not able to determine whether two adjacent nodes belong to the same empty region. Hence, it can only construct the primary and secondary circuits along the cycle. Subsequently, it handles each of its occurrences within the cycle separately. Nonetheless, the adjusted construction still satisfies the necessary properties given in Sect. 3.2.

The second step is the computation of the paths from the starting points straight into direction d_p. The canonical skeleton algorithm proceeds as follows. Each inner amoebot connects all pins of its neighbors in directions d_p and $\rho_s(d_p, 180)$, and each boundary amoebot connects all pins to its neighbors in direction d_p and $\rho_s(d_p, 180)$, respectively (see Fig. 21). Each starting point without a second boundary activates the circuit to its neighbor in direction d_p. Each amoebot that receives a beep is part of a path from the starting point straight into direction d_p. Finally, we obtain the following theorem.

Theorem 4.4 *The canonical skeleton algorithm computes a (canonical) skeleton (path) in $O(\log^2 n)$ rounds w.h.p.*

Proof The preprocessing step requires $O(\log n)$ rounds w.h.p. (see Sect. 2). The first step requires $O(\log^2 n)$ rounds for the computation of global maxima (see Sect. 3.3). The second requires $O(1)$ rounds. Altogether, the canonical skeleton algorithm requires $O(\log^2 n)$ rounds w.h.p. □

4.2 Spanning tree

We now show how a skeleton can be utilized to construct a spanning tree. We assume that we have already computed a (not necessarily canonical) skeleton (see Sect. 4.1). Our spanning tree algorithm consists of two phases (see Fig. 22). We first outline the goal of each phase. In the first phase, we construct a tree spanning all amoebots of the skeleton path. In the second phase, we add the remaining amoebots to the tree.

Now, consider the first phase. We make use of the following lemma.

Lemma 4.5 *Let $G = (V, E)$ be a connected graph. Let $\pi = (v_1, \ldots, v_m)$ be a path in G. Let $V' \subseteq V$ be the set of all amoebots on the path π. Let $\pi(v)$ denote the first edge in π incident to v. Then, $T = (V', E')$ with $E' = \bigcup_{v \in V' \setminus \{v_1\}} \{\pi(v)\}$ is a tree.*

Proof In order to prove that T is a tree, we show that T is cycle-free and connected. Each edge $\pi(v) = (v', v)$ implies that v' appears before v in π. Clearly, this relationship cannot be cyclic.

We prove that T is connected by induction on the path π. The induction base holds trivially for v_1. Suppose that all nodes up to node v_i are connected within T. Consider node v_{i+1}. If it is not the first occurrence of v_{i+1} on the path, then v_{i+1} is already connected by induction hypothesis. Otherwise $\pi(v_{i+1}) = \{v_i, v_{i+1}\} \in E'$. This edge connects v_{i+1} to all nodes up to node v_i since these are connected by induction hypothesis. □

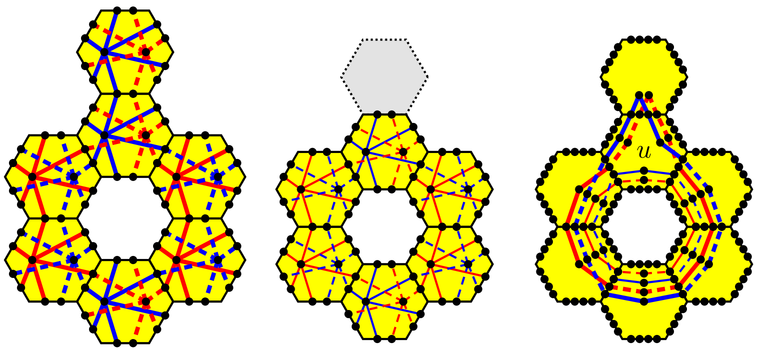
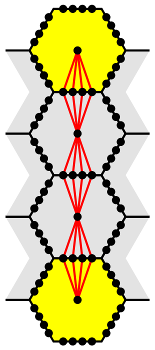


Fig. 20 Computation of the global maxima of each boundary set with respect to $d = N$. The first and second figure show the original construction for the outer and inner boundary set, respectively. The gray amoebot (dotted boundary) does not participate in the computation for the inner boundary set. The third figure shows the construction for the computation in parallel. Amoebot u is unaware that two of its adjacent nodes belong to the same empty region

Fig. 21 Computation of the paths. The figure shows the circuit utilized to identify the paths. The yellow amoebots (top and bottom amoebot) are boundary amoebots. The gray amoebots are inner amoebots



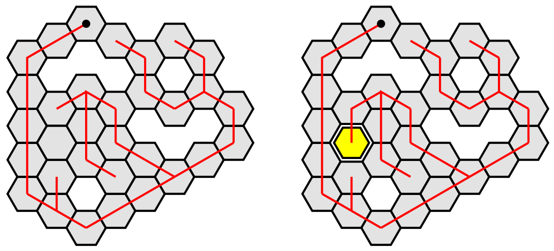


Fig. 22 Spanning tree obtained from the $(N, +)$-skeleton (see Fig. 18). The node indicates the root. The left figure shows the situation after the first phase, and the right figure the situation after the second phase. The yellow amoebot (double boundary) is added to the spanning tree during the second phase

In order to determine the first occurrence of each amoebot, we apply the PASC algorithm algorithm on the path with v_0 as the reference amoebot (see Sect. 3.1). Each amoebot is able to determine its first occurrence by simply comparing the identifiers of all its occurrences. Each amoebot notifies the predecessor of its first occurrence.

Next, consider the second phase. For each amoebot v not included in the skeleton $S \setminus V'$, we add an edge from it to its northern neighbor w to the spanning tree. Note that v is an inner amoebot such that w has to exist. Otherwise, v would be included in the skeleton. Each amoebot $v \in S \setminus V'$ notifies its northern neighbor. We obtain the following theorem.

Theorem 4.6 *Given a skeleton, the spanning tree algorithm computes a spanning tree after $O(\log n)$ rounds. Altogether, it requires $O(\log^2 n)$ rounds w.h.p.*

Proof The correctness follows from Lemma 4.5. The first phase requires $O(\log n)$ rounds (see Sect. 3.1). The second phase requires $O(1)$ rounds. Altogether, the spanning tree algorithm requires $O(\log n)$ rounds. □

Remark 4.7 If the amoebot structure has no holes, we can compute a spanning tree without a skeleton in $O(\log n)$ rounds as follows (see Fig. 23). The algorithm consists of two phases that correspond to the ones of the preceding spanning tree algorithm. In the first phase, we add an edge for each pair of adjacent boundary amoebots with a common unoccupied adjacent node. We obtain a set of cycles that are connected by narrow chains of amoebots. For each cycle, we elect a leader that removes one of its incident edges from the cycle. This takes $O(\log n)$ rounds w.h.p. (Feldmann et al. 2022).

In the second phase, for each inner amoebot, we add an edge from it to its northern neighbor. This takes $O(1)$ rounds.

4.3 Symmetry detection

We now show how to detect rotational symmetries and reflection symmetries. Due to the underlying infinite regular triangular grid graph G_Δ, there is only a limited number of possible symmetries. More precisely, an amoebot structure can only be 2-fold, 3-fold or 6-fold rotationally symmetric, and reflection symmetric to axes in a direction of $D_m \cup D_p$. Moreover, the problem is complicated by the facts that the symmetry point may be an

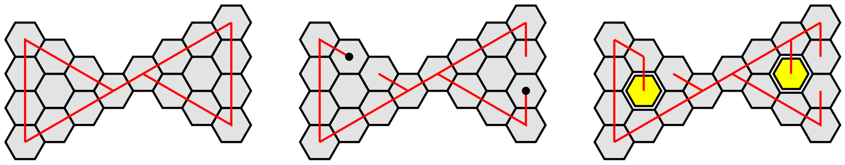


Fig. 23 Spanning tree for amoebot structures without holes. The left and center figure show the situation after the first and second step of the first phase, respectively. The first step results in two cycles connected by a chain of amoebots. The nodes indicate the elected leader. The right figure shows the situation after the second phase. The yellow inner amoebots (double boundary) are added to the spanning tree during this phase

unoccupied node of G_Δ or not a node of G_Δ at all, and that the symmetry axis may not be occupied by any amoebots.

Recall that we define a canonical skeleton by two parameters: the direction d and the sign s. Note that rotating the direction results in a rotated construction, and inverting the sign results in a reflected construction. Hence, a symmetric amoebot structure implies a symmetric construction of canonical skeletons. The idea of our symmetry detection algorithm is therefore to compare the canonical skeletons. We compare the skeletons according to the following observations (compare Fig. 24).

Observation 4.8 *An amoebot structure is 2-fold rotationally symmetric if the $(N, +)$-skeleton and the $(S, +)$-skeleton are symmetric. An amoebot structure is 3-fold rotationally symmetric if the $(N, +)$-skeleton and the $(ESE, +)$-skeleton are symmetric. An amoebot structure is 6-fold rotationally symmetric if it is 2-fold and 3-fold rotationally symmetric.*

Let $d \in D_m \cup D_p$ and let d' denote the direction obtained if we rotate d by $90°$ counterclockwise. An amoebot structure is reflection symmetric to an axis in direction d if the $(d', +)$-skeleton and the $(d', -)$-skeleton are symmetric.

Note that due to symmetry, it is enough to only check half of $D_m \cup D_p$.

In order to compare two canonical skeletons, we map each canonical (d, s)-skeleton path to a unique bit string by having each amoebot on the path store a partial bit string of constant length encoding the direction of its successor relative to direction d and sign s. Consequently, the comparison of two canonical skeletons is reduced to the comparison of the corresponding bit strings of the two skeletons. In the following, we show how such a comparison of two bit strings is possible in polylogarithmic time.

To this end, we consider the *string equality problem*: Let $A = (A_0, \ldots, A_{m-1})$ and $B = (B_0, \ldots, B_{m'-1})$ be two chains of amoebots with reference amoebots A_0 and B_0, holding bit strings $a = (a_0, \ldots, a_{m-1})$ and $b = (b_0, \ldots, b_{m'-1})$. We show how a and b can be checked for equality in time $O(\log^5 m)$ w.h.p. using probabilistic polynomial identity testing.-

We first give a high-level overview of our solution: Since we can compare the length of A and B by comparing the identifiers $\mathrm{id}_{A,A_0}(A_{m-1}) = m - 1$ and $\mathrm{id}_{B,B_0}(B_{m'-1}) = m' - 1$ of the last amoebots of the chains bit by bit with the

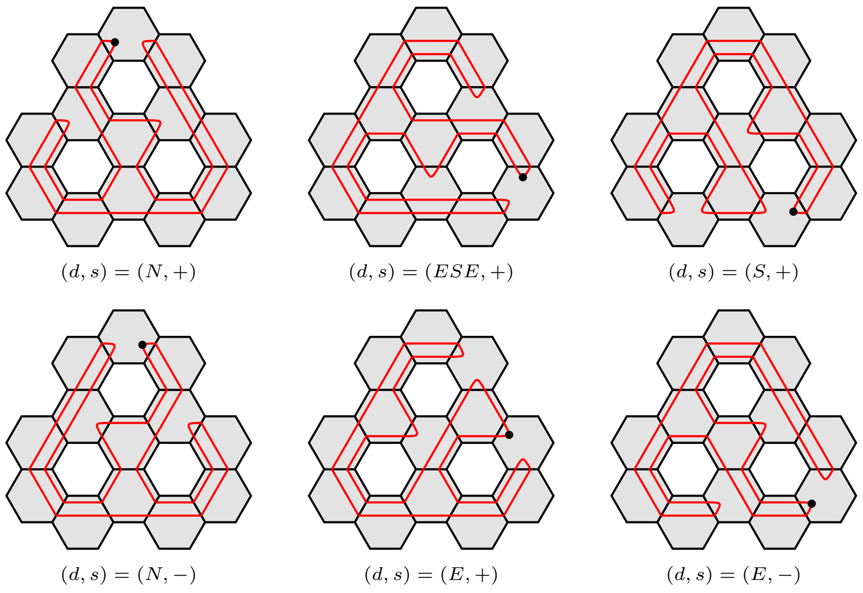


Fig. 24 Symmetries of an amoebot structure. The amoebot structure is 3-fold rotational symmetric since the $(N, +)$–and the $(ESE, +)$-skeleton are symmetric, but not 2-fold or 6-fold rotational symmetric since the $(N, +)$– and the $(S, +)$-skeleton are not symmetric. Further, the amoebot structure is reflection symmetric to an axis into northern direction since the $(N, +)$–and the $(N, -)$-skeleton are symmetric, but not reflection symmetric to an axis into eastern direction since the $(E, +)$–and the $(E, -)$-skeleton are not symmetric

PASC algorithm (see Sect. 3.1) in time $O(\log m)$, we can assume $m = m'$ in the following. Let $c \in \mathbb{N}$. Chain A generates a prime $p \geq 2m$ and repeats the following procedure: A samples r uniformly at random from $[p]$ and sends the pair (p, r) to chain B. Chain A computes $f_a(r) = \sum_{i=0}^{m-1} a_i r^i (\text{mod p})$, and chain B computes $f_b(r) = \sum_{i=0}^{m-1} b_i r^i (\text{mod p})$ and sends the result to chain A which outputs "$a \neq b$" if $f_a(r) \neq f_b(r)$ and repeats the procedure otherwise. After $c\lceil \log m \rceil$ repetitions, A outputs "$a = b$". Note that $a = b$ implies $f_a(r) = f_b(r)$. From the Schwartz-Zippel lemma follows that the one-sided error probability for a single repetition is $\Pr[f_a(r) = f_b(r) \mid a \neq b] \leq m/p \leq 1/2$. It follows $\Pr[A \text{ outputs "}a = b\text{"} \mid a \neq b] \leq 1/m^c$.

We now describe the algorithm in more detail: First, we describe a *block primitive* that we use to divide the chain A into blocks of length $k = O(\log m)$ where $k = 2^{\lceil \log \lambda \rceil}$, $\lambda = 2\,l$ and $l = \lceil \log m \rceil + 2$. Note that $k \geq \lambda$. We have $k \leq m$ for $m \geq 44 =: \eta$. From here on we assume $m \geq \eta$ (in case $m < \eta$, the chains A and B can simply compare their bit strings deterministically). Since the PASC algorithm terminates after $\lceil \log m \rceil$ iterations, we can easily determine amoebot A_λ by using the PASC algorithm 2 times and forwarding a marker after every iteration. Then we use the PASC algorithm again, with the following addition (compare Fig. 25): For an amoebot let Q be the partition set on which it received a beep (either its primary or secondary partition set). An active amoebot (except A_0) splits Q into singletons. We obtain a circuit between each pair of consecutive active amoebots. Then, A_0 beeps on Q. If A_λ receives a beep, the procedure terminates, otherwise we continue with the next iteration. After termination, exactly the amoebots A_{ik} are active. Since we can directly compare the bits a_i of the amoebots between the last active amoebot and A_{m-1} with the corresponding bits b_i of chain B in time $O(\log m)$, we assume in the following w.l.o.g. that $k \mid m$ holds. This enables us to divide the amoebots of A into m/k chains $C_i = (A_{ik}, \ldots, A_{(i+1)k-1})$ of length k with reference amoebot A_{ik}.

Now we describe how A generates a prime $p \geq 2m$. Chain A samples an l-bit integer $p = (p_0, \ldots, p_{l-2}, 1)$ uniformly at random such that A_i stores p_i. Note that the most significant bit is fixed to 1 and therefore $p \in [2^{l-1}, 2^l)$, in particular $2m \leq p < 4m$. We check deterministically whether p is a prime by checking in parallel for all $2 \leq t < m$ whether $t \mid p$ (note that $\lfloor \sqrt{p} \rfloor < m$): First, p and $t = \text{id}_{A,A_0}(A_{ik}) = ik$ (using the PASC algorithm) are stored in the first l amoebots of every chain C_i in time $O(\log m)$. Then, all chains C_i repeat the following procedure in parallel for at most k times: If $t \geq 2$, check whether $t \mid p$ in time $O(\log^2 m)$ using binary long division with remainder. Abort the prime testing, if $t \mid p$, otherwise increment t.

We repeat the entire procedure at most $3cl^2$ times or until we have successfully sampled a prime p. The runtime for the prime generation is $O(\log^5 m)$. We now analyse the probability for the event E_{fail} that no prime is generated. Using non-asymptotic bounds on the prime-counting function, one can show that the fraction of integers in $[2^{l-1}, 2^l)$ that are prime is at least $1/(3l)$. It follows: $\Pr[E_{\text{fail}}] \leq (1 - 1/(3\,l))^{3cl^2} \leq 1/e^{cl} \leq 1/m^c$

We now address the probabilistic polynomial identity testing, focusing on the computation of $f_a(r)$ for $r \in [p]$. We use the previously determined division of the chain A into blocks of length $k = O(\log m)$. Assume that p, r and $e = \text{id}_{A,A_0}(A_{ik}) = ik$ are stored in the first l amoebots of every chain C_i. All chains C_i repeat the following procedure in parallel for k times: Compute $s^{(i)} = a_e r^e (\text{mod p})$ using modular exponentiation via the right-to-left binary method. Using binary long multiplication and division with remainder, this step is possible in time $O(\log^3 m)$. Then, increment e. Once all chains C_i have completed the j-th repetition, we compute the sum of the $s^{(i)}$ modulo p and store it in the first l amoebots of chain A using a generalization of Theorem 2.2. The summation is possible in time $O(\log^2 m)$. The computed sum is then added to a running total modulo p.

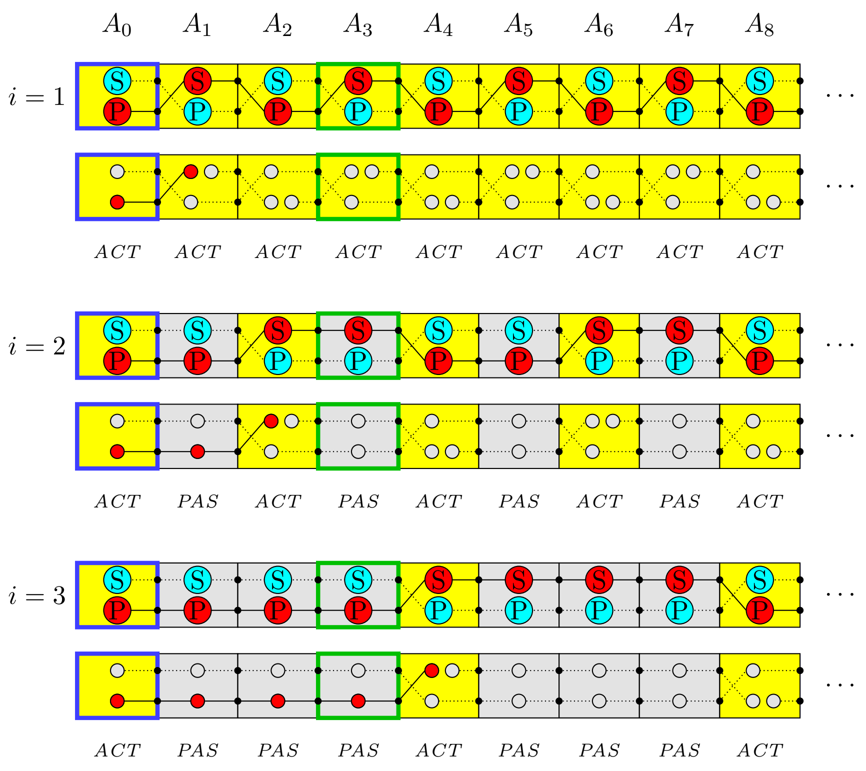
Finally, after k repetitions, the result $f_a(r)$ is stored in the amoebots $A_0, \ldots, A_{l-1}$. The runtime for the polynomial identity testing is $O(\log^4 m)$.

Note that the size of the outer boundary set is $\Omega(\sqrt{n})$, which is also a lower bound for the size of a canonical skeleton. Hence, we get the following result:

Theorem 4.9 *The string equality problem on chains of length $O(m)$ can be solved in $O(\log^5 m)$ rounds w.h.p. Therefore, the symmetry detection problem can be solved in $O(\log^5 n)$ rounds w.h.p.*

Finally, we discuss how to compute the amoebot occupying the symmetry point and amoebots on the symmetry axis, respectively, if such exist. The idea is to identify some symmetric amoebots, to compute symmetric identifiers with these as reference amoebots, and to output all amoebots that receive the same identifier for each reference amoebot. In the following, we discuss all cases of symmetry in more detail.

First, consider an amoebot structure that is reflection symmetric to an axis into direction d (see Fig. 26). Let d_1 and d_2 be the two to d perpendicular directions, i.e., the directions obtained if we rotate d by 90° clockwise and counterclockwise, respectively. We proceed as follows. We first compute the global maxima of S with respect to d_1 and d_2, respectively. Let S_{d_1} and S_{d_2}, respectively, denote these global maxima. Then, we compute the global maximum of



A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
$i = 1$
$i = 2$
$i = 3$
S
P
ACT ACT ACT ACT ACT ACT ACT ACT ACT
ACT PAS ACT PAS ACT PAS ACT PAS ACT
ACT PAS PAS PAS ACT PAS PAS PAS ACT

Fig. 25 Block primitive. The figure shows how a chain A with reference amoebot A_0 and a marked amoebot A_λ can be divided into blocks of length $k \geq \lambda$ where $k = 2^{\lceil \log \lambda \rceil}$ using the PASC algorithm (compare Fig. 5) with an additional step after each iteration. Here, we have $\lambda = 3$ and $k = 4$. The first line shows the chain at the beginning of the i-th iteration of the PASC algorithm. The circles indicate the partition sets. The blue bordered amoebot is A_0, and the green bordered amoebot is $A_\lambda = A_3$. Yellow amoebots are active, and gray amoebots are passive. The second line shows the configuration resulting from the following additional step: For an amoebot let Q be the partition set on which it received a beep (depicted in red; either its primary or secondary partition set). An active amoebot (except A_0) splits Q into singletons. We obtain a circuit between each pair of consecutive active amoebots. Then, A_0 beeps on Q. If A_λ receives a beep, the procedure terminates, otherwise continue with the next iteration. After termination, exactly the amoebots A_{ik} are active

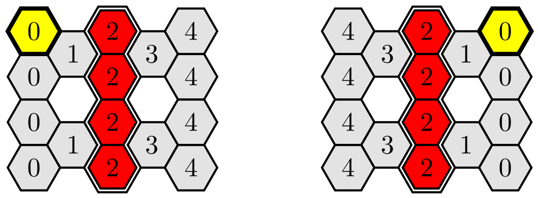


Fig. 26 Amoebot structure reflection symmetric to an axis into direction $d = N$. The yellow amoebots (thick boundary) indicate amoebots u_1 and u_2. The red amoebots (double boundary) are on the symmetry axis. Note that each red amoebot receives two equal identifiers, i.e., it receives the identifier 2 twice

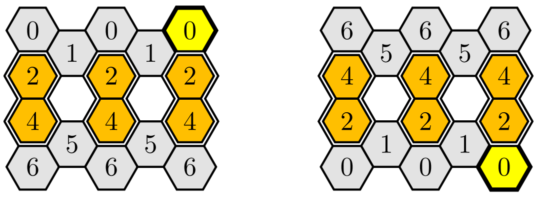


Fig. 27 Amoebot structure reflection symmetric to an axis into direction $d = E$. The yellow amoebots (thick boundary) indicate amoebots u_1 and u_2. The edge between each pair of adjacent orange amoebots (double boundary) intersects the symmetry axis. Note that each orange amoebot of a pair receives a different permutation of the same tuple of identifiers, i.e., the northern amoebot receives the tuple (2, 4), and the southern amoebot receives the tuple (4, 2)

S_{d_1} and S_{d_2}, respectively, with respect to direction d. Let u_1 and u_2, respectively, denote the global maximum. Now, we compute a 2-tuple. It contains the spatial identifiers for u_1 and d_2, and for u_2 and d_1. Each amoebot on the symmetry axis receives two equal identifiers. These identifiers are equal for each amoebot on the symmetry axis. Note that there may be no amoebot on the symmetry axis if $d \in D_p$ (see Fig. 27). The symmetry axis intersects an edge of G_S if both incident amoebots receive different permutations of the same tuple of identifiers.

Second, consider an amoebot structure that is 2-fold rotational symmetric (see Fig. 28). We proceed as follows. We first compute the northernmost amoebot of the westernmost amoebots, and the southernmost amoebot of the easternmost amoebots, respectively. Let u_1 denote the

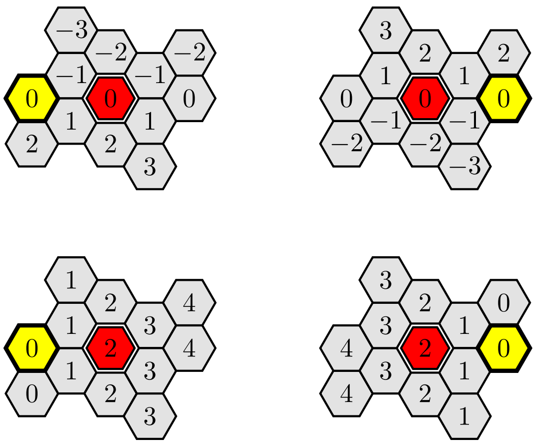


Fig. 28 Amoebot structure 2-fold rotational symmetric to a node of G_Δ. The yellow amoebots (thick boundary) indicate amoebots u_1 and u_2. The upper and bottom figures shows the first and second tuple of spatial identifiers, respectively. The red amoebot (double boundary) lies on the symmetry point. Note that the red amoebot receives two equal identifiers in both tuples, respectively, i.e., it receives the identifier 0 twice in the first tuple, and identifier 2 twice in the second tuple

former, and u_2 the latter. Next, we compute two 2-tuples. The first tuple contains the spatial identifiers for u_1 and the southern direction, and for u_2 and the northern direction. The second tuple contains the spatial identifiers for u_1 and the eastern direction, and for u_2 and the western direction.

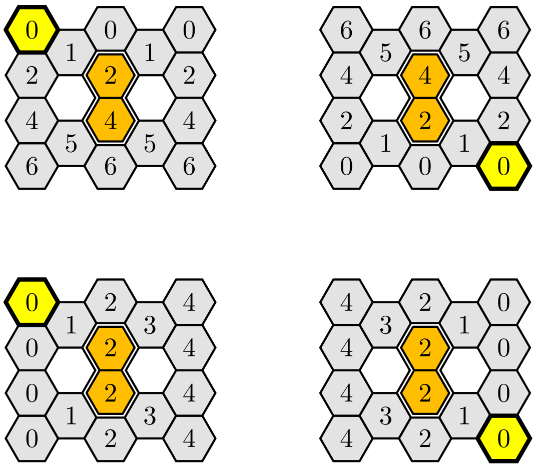


Fig. 29 Amoebot structure 2-fold rotational symmetric to a point on an edge of G_Δ. The yellow amoebots (thick boundary) indicate amoebots u_1 and u_2. The upper and bottom figures shows the first and second tuple of spatial identifiers, respectively. The symmetry point lies on the edge between the two orange amoebots (double boundary). Note that each orange amoebot receives different permutations of the same tuples of identifiers, i.e., the northern amoebot receives the tuples (2, 4) and (2, 2), and the southern amoebot receives the tuple (4, 2) and (2, 2)

An amoebot occupies the symmetry point iff it receives two equal identifiers in both tuples, respectively. Note that the symmetry point may be unoccupied. Furthermore, the symmetry point may lie between two amoebots (see Fig. 29). In this case, both amoebots receive different permutations of the same tuples of identifiers.

Third, consider an amoebot structure that is 3-fold rotational symmetric (see Fig. 30). We proceed as follows. Analogously to the case of 2-fold rotational symmetry, we first compute three symmetric amoebots. (More precisely, we compute the westernmost amoebot of the northernmost amoebots, the south-southeasternmost amoebot of the west-southwesternmost amoebots, and the north-northeastern amoebot of the east-southeasternmost amoebots.) Then, we compute a 3-tuple. The tuple contains identifiers equal to the distances in G_Δ for each of these amoebots. An amoebot occupies the symmetry point iff it receives three equal identifiers. Note that the symmetry point may be unoccupied. Furthermore, the symmetry point may lie between three amoebots (see Fig. 31). In this case, all amoebots receive different permutations of the same tuple of identifiers.

Naturally, for amoebot structures that are 6-fold rotationally symmetric, we can use both the procedure for 2-fold and 3-fold rotationally symmetric amoebot structures. We obtain the following result.

Proposition 4.10 *By applying the global maxima and PASC algorithm, we can compute the amoebot occupying the symmetry point and amoebots on the symmetry axis, respectively, within* $O(\log^2 n)$ *rounds w.h.p. if such exist.*

5 Conclusion and future work

In this paper, we have proposed polylogarithmic-time solutions for a range of problems. First, we have computed spatial identifiers in order to compute a stripe through a given amoebot and direction, and the global maxima of the given amoebot structure with respect to a direction. Using these results, we have constructed a canonical skeleton path, which provides a unique characterization of the shape of the given amoebot structure. Constructing canonical skeleton paths for different directions will then allow the amoebots to set up a spanning tree and to check symmetry properties of the given amoebot structure.

Our solutions could be useful for various problems like rapid shape transformation, energy dissemination, and structural monitoring (see Sect. 1). However, the details still have to be worked out, including appropriate models and assumptions. Beyond that, we think that exploring further applications for the spatial identifiers and the skeleton would be interesting.

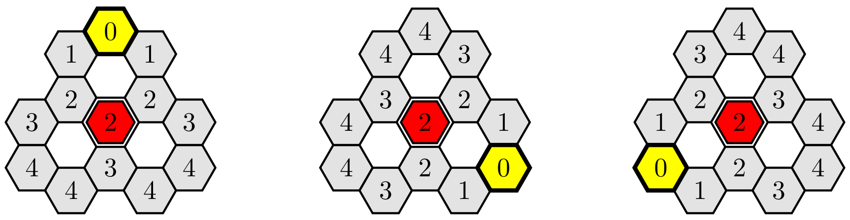


Fig. 30 Amoebot structure 3-fold rotational symmetric to a node of G_Δ. The yellow amoebots (thick boundary) indicate the three symmetric amoebots. The red amoebot (double boundary) lies on the symmetry point. Note that each red amoebot receives three equal identifiers, i.e., it receives the identifier 2 three times

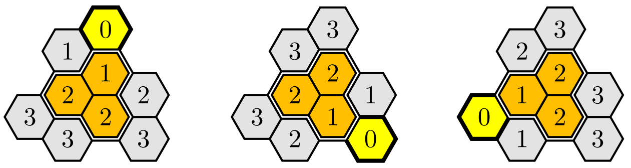


Fig. 31 Amoebot structure 3-fold rotational symmetric to a point within a face of G_Δ. The yellow amoebots (thick boundary) indicate the three symmetric amoebots. The symmetry point lies within the triangle between the three orange amoebots (double boundary). Note that each orange amoebot receives a different permutation of the same tuple of identifiers, i.e., the northern amoebot receives the tuple (1, 2, 2), the southern amoebot receives the tuple (2, 1, 2), and the western amoebot the tuple (2, 2, 1)

Author Contributions Andreas Padalkin wrote the main manuscript text and prepared all figures. Daniel Warner wrote the part about the string equality problem in Sect. 4.3. All authors reviewed the manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. This work has been supported by the DFG Project SCHE 1592/6-1 (PROGMATTER).

Data Availability We do not analyse or generate any datasets, because our work proceeds within a theoretical and mathematical approach.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Ethical Approval Not applicable.

References

Alumbaugh JC, Daymude JJ, Demaine ED, et al (2019) Simulation of programmable matter systems using active tile-based self-assembly. In: DNA, lecture notes in computer science, vol 11648. Springer, pp 140–158. https://doi.org/10.1007/978-3-030-26807-7_8

Arroyo MA, Cannon S, Daymude JJ et al (2018) A stochastic approach to shortcut bridging in programmable matter. Nat Comput 17(4):723–741

Cannon S, Daymude JJ, Randall D, et al (2016) A markov chain algorithm for compression in self-organizing particle systems. In: PODC. ACM, pp 279–288

Daymude JJ, Hinnenthal K, Richa AW, et al (2019) Computing by programmable particles. In: Distributed computing by mobile entities, lecture notes in computer science, vol 11340. Springer, pp 615–681, https://doi.org/10.1007/978-3-030-11072-7_22

Daymude JJ, Gmyr R, Hinnenthal K, et al (2020) Convex hull formation for programmable matter. In: ICDCN. ACM, pp 2:1–2:10, https://doi.org/10.1145/3369740.3372916

Daymude JJ, Richa AW, Weber JW (2021) Bio-inspired energy distribution for programmable matter. In: ICDCN. ACM, pp 86–95. https://doi.org/10.1145/3427796.3427835

Daymude JJ, Richa AW, Scheideler C (2023) The canonical amoebot model: algorithms and concurrency control. Distributed Comput 36(2):159–192

Derakhshandeh Z, Dolev S, Gmyr R, et al (2014) Brief announcement: amoebot - a new model for programmable matter. In: SPAA. ACM, pp 220–222, https://doi.org/10.1145/2612669.2612712

Derakhshandeh Z, Gmyr R, Strothmann T, et al (2015) Leader election and shape formation with self-organizing programmable matter. In: DNA, Lecture Notes in Computer Science, vol 9211. Springer, pp 117–132, https://doi.org/10.1007/978-3-319-21999-8_8

Derakhshandeh Z, Gmyr R, Richa AW, et al (2016) Universal shape formation for programmable matter. In: SPAA. ACM, pp 289–299. https://doi.org/10.1145/2935764.2935784

Derakhshandeh Z, Gmyr R, Richa AW et al (2017) Universal coating for programmable matter. Theor Comput Sci 671:56–68. https://doi.org/10.1016/j.tcs.2016.02.039

Feldmann M, Padalkin A, Scheideler C et al (2022) Coordinating amoebots via reconfigurable circuits. J Comput Biol 29(4):317–343. https://doi.org/10.1089/cmb.2021.0363

Luna GAD, Flocchini P, Santoro N et al (2020) Shape formation by programmable particles. Distributed Comput 33(1):69–101. https://doi.org/10.1007/s00446-019-00350-6

Padilla JE, Sha R, Kristiansen M et al (2015) A signal-passing dna-strand-exchange mechanism for active self-assembly of dna nanostructures. Angew Chem Int Ed 54(20):5939–5942. https://doi.org/10.1002/ange.201500252

Pandurangan G, Robinson P, Scquizzato M (2018) The distributed minimum spanning tree problem. Bull EATCS 125

Scalise D, Schulman R (2019) Controlling matter at the molecular scale with dna circuits. Annu Rev Biomed Eng 21(1):469–493. https://doi.org/10.1146/annurev-bioeng-060418-052357

Shah S, Wee J, Song T et al (2020) Using strand displacing polymerase to program chemical reaction networks. J Am Chem Soc 142(21):9587–9593. https://doi.org/10.1021/jacs.0c02240

Song T, Eshra A, Shah S et al (2019) Fast and compact dna logic circuits based on single-stranded gates using strand-displacing polymerase. Nat Nanotechnol 14(11):1075–1081. https://doi.org/10.1038/s41565-019-0544-5

Toffoli T, Margolus N (1993) Programmable matter: Concepts and realization. Int J High Speed Comput 5(2):155–170. https://doi.org/10.1142/S0129053393000086