Theoretical Computer Science 1057 (2025) 115553

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Optimal gathering of robots in anonymous butterfly networks via leader election

Serafino Cicerone [a,*], Alessia Di Fonso [a], Gabriele Di Stefano [a], Alfredo Navarra [b]

[a] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Universitàdegli Studi dell'Aquila, I-67100, L'Aquila, Italy
[b] Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, I-06123, Perugia, Italy

ARTICLE INFO

Section Editor: Pinyan Lu
Handling Editor: Katie Harris

Keywords:
Mobile robots
Synchrony
Gathering
Butterfly graphs
Time complexity
Leader election

ABSTRACT

Robots with very weak capabilities placed on the vertices of a graph are required to move toward a common vertex from where they do not move anymore. The task is known as the *Gathering* problem and it has been extensively studied in the last decade with respect to both general graphs and specific topologies. Most of the challenges faced are due to possible isometries observable from the placement of the robots with respect to the underlying topology. Rings, Grids, and Complete graphs are just a few examples of very regular topologies where the placement of the robots and suitable movements are crucial for succeeding in Gathering. Here we are interested in understanding what can be done in Butterfly graphs where really many isometries are present and most importantly unavoidable by any movement. We propose a Gathering algorithm for the so-called *leader* configurations, i.e., those where the initial placement of the robots admits the detection (and election) of one robot as the leader. We introduce a non-trivial technique to elect the leader which is of its own interest. We also prove that the proposed Gathering algorithm is asymptotically optimal in terms of synchronous rounds required.

1. Introduction

A swarm of very weak - in terms of capabilities - robots, placed on the vertices of an anonymous graph, are required to move toward a common vertex (not known in advance) from where they do not move anymore. Initially, each vertex is occupied by at most one robot. The task is known in the literature as the *Gathering* and it has been extensively studied in the last decade for many different assumptions on the robots' capabilities. The common theoretical approach is to detect the minimal capabilities required.

In this paper, we consider the $\mathcal{OBLOT}$ model [26,28], where synchronized robots are assumed to be: *Anonymous*: no unique identifiers; *Autonomous*: no centralized control; *Dimensionless*: no occupancy constraints, no volume; *Oblivious*: no memory of past events; *Homogeneous*: they all execute the same *deterministic* algorithm; *Silent*: no means of direct communication; *Disoriented*: no common coordinate system, nor left-right orientation.

Each robot continuously executes a `Look-Compute-Move` (`LCM`) cycle by performing the following three operations in sequence:

- `Look`: The robot observes the environment. The result of this phase is a snapshot of the positions of all robots with respect to the local perception (and not to a global reference system);

* Corresponding author.
E-mail addresses: serafino.cicerone@univaq.it (S. Cicerone), alessia.difonso@univaq.it (A. Di Fonso), gabriele.distefano@univaq.it (G. Di Stefano), alfredo.navarra@unipg.it (A. Navarra).

https://doi.org/10.1016/j.tcs.2025.115553
Received 17 August 2024; Received in revised form 28 July 2025; Accepted 8 September 2025
Available online 11 September 2025
0304-3975/© 2025 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Table 1
Achievements about gathering on graphs in $\mathcal{OBLOT}$. The last result marked by (*) is achieved in this paper.

Graph	Ref.	Schedule	Multiplicity	Achievement
Any	[23]	FSync	global strong	some impossibility
	[23]	Async	global strong	basic feasibility
	[10]	SSync	global strong	sufficient condition
Trees	[15,16]	Async	none	full characterization
	[23]	Async	global strong	traveled distance opt
Rings	[34]	Async	global weak	asymmetric conf.s
	[33]	Async	global weak	symmetric dense conf.s
	[18]	Async	global weak	conf.s with 6 robots
	[2]	Async	global weak	some conf.s with 4 robots
	[31]	Async	local weak	opt asymmetric
	[17]	Async	global weak	(almost) full characterization
	[19]	Async	local weak	asymmetric conf.s
	[20]	Async	local weak	full characterization
	[23]	Async	global strong	traveled distance opt
	[21]	SSync	global weak	separation with Async
Regular Bipartite	[29]	Async	local weak	full characterization
Finite Grids	[15]	Async	none	full characterization
Infinite Grids	[22]	Async	global strong	traveled distance opt
Tori	[32]	Async	local weak	asymmetric conf.s
Oriented Hypercubes	[3]	Async	global weak	traveled distance opt
Complete	[10]	FSync	global strong	opt full characterization
	[9]	SSync	global strong	impossibility
Complete Bipartite	[10]	FSync	global strong	full characterization
	[9]	SSync	global strong	some conf.s
Butterflies	(*)	FSync	global weak	leader conf.s time opt

- `Compute`: The robot executes the designed algorithm to determine its next move, using only the data sensed in the `Look` phase as input. For robots moving in graphs, the result of this phase either is the vertex where the robot currently resides or it is a vertex among those in the direct neighborhood (i.e., at most one edge per cycle can be traversed);
- `Move`: The robot moves toward the computed target. If the target is the current position, then the robot performs what is called a *nil* movement.

The time can be logically divided into global rounds. In each round, all the robots perform a `LCM` cycle. If $C(t)$ denotes the configuration observed by the robots at time t during their `Look` phase, then an *execution* of an algorithm $\mathcal{A}$ from an initial configuration C is a sequence of configurations $\mathbb{E} : C(0), C(1), \ldots$, where $C(0) = C$ and $C(t+1)$ is obtained from $C(t)$ by moving each robot according to the result of the `Compute` phase as implemented by $\mathcal{A}$. On graphs, moves are always considered as *instantaneous*. This results in always perceiving robots on vertices and never on edges during `Look` phases. The *time complexity* of an algorithm $\mathcal{A}$ is the maximum amount of rounds required by $\mathcal{A}$ for processing any initial configuration.

It is very common (as dictated by impossibility results) that in combination with the `LCM`-model, robots are endowed with the so-called *multiplicity detection* capability (see e.g. [13,34]). Basically, when more than one robot resides on the same vertex x, then x is said to be occupied by a *multiplicity*. During the `Look` phase, robots can perceive *multiplicities*. The multiplicity detection capability might be *local* or *global*, depending on whether the multiplicity is detected only by robots composing the multiplicity itself or by any robot performing the `Look` phase, respectively. Moreover, the multiplicity detection can be *weak* or *strong*, depending on whether a robot can detect only the presence of a multiplicity or if it perceives the exact number of robots composing the multiplicity, respectively.

Related work. The Gathering problem has been originally studied in the context of robots freely moving on the Euclidean plane, under different settings [14,27]. Gathering is a special case of the more general *pattern formation* problem [6–8,11,38]. Indeed, it is also known as *Point formation*. Gathering has been fully characterized in [13] for robots endowed with global weak multiplicity detection.

In contrast, for robots constrained to move on graphs, it turns out to be very difficult to envision general results as different challenges arise for different topologies. In fact, apart from some impossibility results or basic conditions that guarantee the resolution of the Gathering task provided in [10,23], most of the literature focuses on specific topologies. Table 1 summarizes the results achieved so far, with respect to the Gathering task in different topologies.

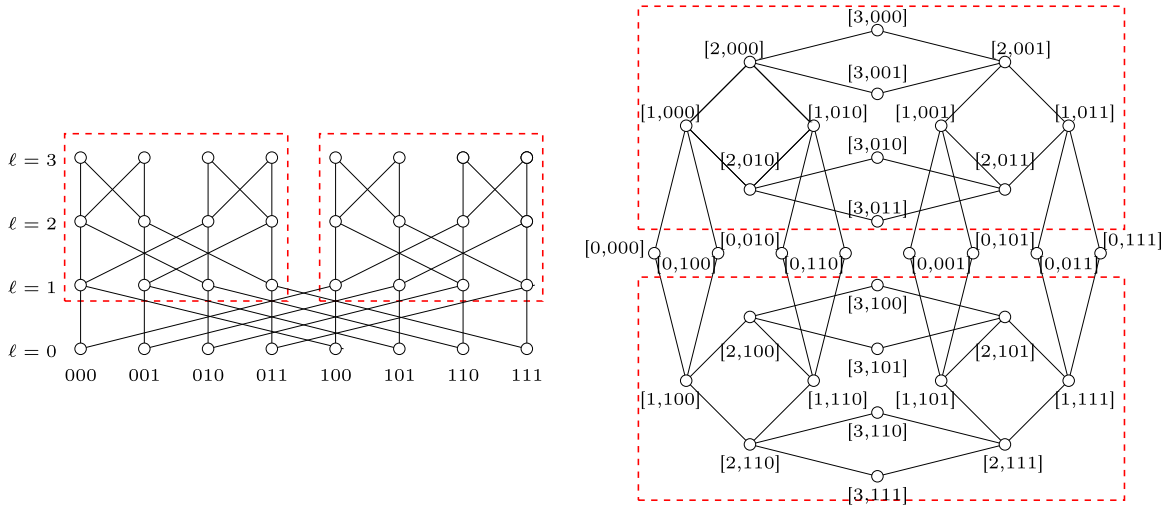


Fig. 1. Normal representation and diamond representation of $BF(3)$. The dashed rectangles contains two copies of $BF(2)$ that, along with 2^3 additional vertices for layer 0, form $BF(3)$.

Most of these topologies are very symmetric, that is, vertices can be partitioned into a few classes of equivalence. The choice has been dictated by the requirement to make the design of a resolution algorithm harder as robots cannot exploit many topological properties. More recent studies on Gathering concern fault-tolerance [1,4,24,25,30,37] or dynamic networks [35].

Our results. In this paper, we focus on synchronous robots endowed with global weak multiplicity detection, moving on Butterfly graphs.

Moving along Butterflies turns out to be really difficult as such graphs admit a lot of isometries. Moreover, even when some robots are distinguishable because of their positioning with respect to other robots, still some vertices remain *equivalent*, i.e., indistinguishable. From [23, Theorem 2], we know that on *partitive* configurations, gathering is unsolvable. Intuitively, a configuration is said to be partitive if its vertices can be partitioned into subsets of the same cardinality greater than one, where each subset contains equivalent vertices also with respect to the occupancy of the robots. In this paper, we propose a Gathering algorithm for the so-called *leader* configurations, i.e., those where the initial placement of the robots admits to detect (and elect) one robot as the leader. We introduce a non-trivial technique to elect the leader which is of its own interest. We also prove that the proposed Gathering algorithm is asymptotically optimal in terms of rounds required.

2. Notation and basic concepts

A d-dimensional butterfly $BF(d)$ (e.g., see [36]) is an undirected graph with vertices $[\ell, c]$, where $\ell \in \{0, 1, \ldots, d\}$ is the *layer* and $c \in \{0,1\}^d$ is the *column*. Note that when $d = 0$ the set $\{0,1\}^d$ is a singleton set consisting only of the null (empty) bit string. The vertices $[\ell, c]$ and $[\ell', c']$ are adjacent if $|\ell - \ell'| = 1$, and either $c = c'$ or c and c' differ precisely in the ℓ-th bit. $BF(d)$ has $d+1$ layers with 2^d vertices at each layer, which gives $(d+1) \cdot 2^d$ vertices in total. The vertices at layer 0 and d have degree 2 whereas the rest of the vertices have degree 4. We call layer 0 and d *boundary layers*. $BF(d)$ has two standard graphical representations, namely normal and diamond, see 1. Note that, $BF(d)$ can be also recursively built by using two copies of $BF(d-1)$ along with 2^d additional vertices, those forming layer 0 (or layer d). This observation leads to the following useful property (cf. diamond representation in 1):

property 1. Let V_1 and V_2 be the sets of vertices of the two copies of $BF(d-1)$ that, along with vertices at layer 0 (layer d, resp.), form $BF(d)$. If $u \in V_1$ and $v \in V_2$, then each u,v-shortest path within $BF(d)$ contains a vertex at layer 0 (layer d, resp.).

Configurations. We consider configurations of robots moving in a d-dimensional butterfly. As robots are assumed to be endowed with global weak multiplicity detection, a configuration is modeled as $C = (BF(d), \lambda)$, where $\lambda : V \to \{0, 1, M\}$ is a function that specifies if each vertex is *unoccupied*, *occupied* by one robot or by more than one robot (that is a multiplicity), respectively.

When the Gathering problem is concerned, two additional concepts are useful: C is *initial* if $\lambda(v) \in \{0,1\}$ for each $v \in V$ (each robot lies on a different vertex) and $\sum_{v \in V} \lambda(v) \geq 2$ (two or more robots are required otherwise the problem is trivially solved); [1] it is *final* if $\exists u \in V : \lambda(u) = M$ and $\lambda(v) = 0,\ \forall v \in V \setminus \{u\}$ (all the robots are on a single vertex). A *gathering algorithm* for this problem is a deterministic distributed algorithm that, starting from any initial configuration and regardless of the adversary[2], brings all the

[1] We remark that the definition of "initial configuration" is used to specify all the configurations without multiplicities. Such configurations are those provided as input to the algorithm, but they may also appear during its execution.

[2] As usual in $\mathcal{OBLOT}$, the role of the adversary basically determines the worst case scenario. In our context, this will be better clarified in the subsequent sections.

robots to a final configuration in a finite number of LCM-cycles, i.e., in a finite number of *rounds*. Formally, an algorithm $\mathcal{A}$ solves the gathering problem for an *initial configuration* C if, for any execution $\mathbb{E} : C{=}C(0), C(1), C(2), \ldots$ of $\mathcal{A}$, there exists a time instant $t > 0$ such that $C(t)$ is final and no robots move after t, i.e., $C(t') = C(t)$ holds for all $t' \geq t$. Given an initial configuration $C = (G, \lambda)$, it is worth remarking that many different placements of robots correspond to C due to possible multiplicities. If there exists a gathering algorithm for all the placements of robots corresponding to C, we say that C is *gatherable*, otherwise we say that C is *ungatherable*.

Note that any gathering algorithm cannot exploit the labeling shown in Fig. 1, but can only use topological properties and function λ.

Automorphisms and ungatherable configurations. Two undirected graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there is a bijection φ from V to V' such that $\{u, v\} \in E$ if and only if $\{\varphi(u), \varphi(v)\} \in E'$. An *automorphism* on a graph G is an isomorphism from G to itself. The set of automorphisms of a given graph, under the composition operation, forms the automorphism group of the graph denoted by Aut(G). Two vertices u and v of G are *equivalent vertices* if there exists an automorphism $\varphi \in \text{Aut}(G)$ such that $\varphi(u) = v$. The equivalence classes of the vertices of G under the action of the automorphisms are called vertex *orbits*. Concerning butterfly graphs, it is easy to observe that each orbit O_i of $BF(d)$ is composed of *complementary* layers i and $d - i$, with $0 \leq i \leq \lfloor d/2 \rfloor$. By *intermediate orbit* we mean $O_{d/2}$ when d is even. Notice that the intermediate orbit is formed by one layer only while the other orbits are formed by two layers each; moreover, O_0 corresponds to the union of the boundary layers.

The concept of isomorphism can be easily extended to configurations. In our context, $(BF(d), \lambda)$ and $(BF'(d), \lambda')$ are isomorphic if there exists $\varphi \in \text{Aut}(BF(d))$ such that $\lambda(v) = \lambda'(\varphi(v))$ for each vertex v. As for graphs, Aut(C) is the automorphism group of $C = (BF(d), \lambda)$. In C, two robots r and r' are *equivalent robots* if they reside on vertices that are equivalent according to some automorphism of Aut(C). In what follows, with the notation O_i, $0 \leq i \leq \lfloor d/2 \rfloor$ we always refer to an orbit of $BF(d)$, not to an orbit of a configuration. Note that, for $d > 0$ each orbit O_i has an even number of vertices.

Observation 1. *Let $C = (BF(d), \lambda)$ be an initial configuration. A necessary condition for two robots r and r' to be equivalent in C is that these robots are located at the same orbit O_i, for some $0 \leq i \leq \lfloor d/2 \rfloor$.*

From an algorithmic point of view, it is important to remark that when there are equivalent vertices in a configuration, a robot r located at a vertex u cannot distinguish its position from that of a robot r' located at vertex $v = \varphi(u)$. As a consequence, no algorithm can distinguish between two equivalent robots, and then it cannot avoid that the two robots perform the same move simultaneously. Moreover, if equivalent robots are moved to any orbit of C containing more than one vertex, then no algorithm can specify the destination vertices exactly (in this case, being the destination vertices pairwise equivalent, we assume that the actual destinations are chosen by an adversary).

A configuration is said to be *partitive* if it admits an automorphism such that the vertices can be partitioned into subsets of equal size greater than one, with equivalent vertices within each subset. From [23, Theorem 2], we know that any partitive configuration is ungatherable. To classify all the initial configurations, we introduce the following novel concept.

Definition 1. Let $C = (BF(d), \lambda)$ be any configuration. A robot r located on a vertex v of $BF(d)$ is called *potential-leader* (p-leader for short) if $\lambda(v) = 1$ and $\varphi(v) = v$ for each $\varphi \in \text{Aut}(C)$.

According to Observation 1, if O_i contains just one robot r, then r is a p-leader. We call *leader configuration* any initial configuration C containing at least one p-leader. In what follows, we show that only in leader configurations the robots can elect a single leader.

3. Leader election

In this section, we provide an algorithm that robots can use to elect one of them as a leader in any initial configuration admitting p-leaders. The algorithm is based on an embedding of the butterfly on the Euclidean plane. The embedding is unique up to isomorphisms.

Embedding of a butterfly. The embedding of $BF(d)$ in the Euclidean plane starts from a *root* vertex s and then proceeds level by level, where all the vertices at distance i from s in $BF(d)$ are embedded at the i-th level. In this way, we introduce a sort of *rooted representation* of $BF(d)$. We first describe Hang (cf. 1), the algorithm that embeds $BF(d)$ by using a vertex of degree two (i.e., a vertex $s \in O_0$) as root. Then, we show how this algorithm can be adapted to embed the graph using a vertex of degree four (i.e., a vertex $s \notin O_0$) as root. The algorithm uses the following notation: for each vertex v in the butterfly, v_ℓ and v_c represent the layer and the column, respectively, of v (hence $v = [v_\ell, v_c]$). Given a non-negative integer i, by $b_d(i)$ we denote the binary representation of i with d binary digits.

Algorithm Hang takes the dimension d of the butterfly as input. The coordinates of the vertices of the graph are defined so that a butterfly of dimension d is built by making two copies of a butterfly of dimension $d - 1$, one having positive x coordinates and the other with negative x coordinates. A space of width 2^d is left between the two copies of the butterflies to place the additional vertices of layer d. Let V be the set of vertices of the graph and let X and Y be the sets of x- and y-coordinates of the vertices in V. The algorithm is recursive. The base case (see Lines 2 and 4) defines a butterfly of dimension 0, with a single vertex having layer 0 and an empty string as a column. Both of its Cartesian coordinates are set to 0. At (6), the function is invoked recursively over the dimension $d - 1$. The call returns the set V of the vertices of the graph $BF(d - 1)$ along with their coordinates X and Y. The algorithm builds two copies V_1 and V_2 of the vertices of the butterfly of dimension $d - 1$. Let v_c be the column associated to the vertices in V, the algorithm updates the column of the vertices in V_1, V_2 by concatenating to $\}\}0\varepsilon$, $\}\}1\varepsilon$ to v_c respectively, see 9 and 12. Then, the x coordinates of the vertices in V_1 (V_2, resp.) are shifted to the right (left, resp.) by 2^{d+1} with respect to the x coordinates of the vertices in V. Let V_3 be the set of 2^d vertices of layer d of the butterfly $BF(d)$. These vertices are placed between the two butterflies of dimension $d - 1$ and evenly distributed horizontally over an interval of 2^d, half with positive, half with negative x coordinates, see 17.

Algorithm 1: Hang(d).

```
1  if d = 0 then
2      V = {[0,'']};
3      X([0,'']) = 0;
4      Y([0,'']) = 0;
5      return V, X, Y ;
6  V, X, Y = Hang(d − 1);
7  V_1, V_2, V_3 = ∅
8  foreach v = [v_ℓ, v_c] ∈ V do
9      insert v_1 = [v_ℓ, v_c+"0"] into V_1;
10     X'(v_1) = X(v) + 2^{d+1};
11     Y'(v_1) = Y(v);
12     insert v_2 = [v_ℓ, v_c+"1"] into V_2;
13     X'(v_2) = X(v) − 2^{d+1};
14     Y'(v_2) = Y(v);
15 foreach i ∈ {0, 2^d − 1} do
16     insert v = [d, b_d(i)] into V_3 ;
17     X'(v) = −(2^{d−1} − 1/2) + i ;
18 Y'([d, b_d(0)]) = d;
19 foreach j ∈ {0, d − 1} do
20     foreach i ∈ {2^j, 2^{j+1} − 1} do
21         Y'([d, b_d(i)]) = d − 2(j + 1);
22 return V_1 ∪ V_2 ∪ V_3, X', Y' ;
```

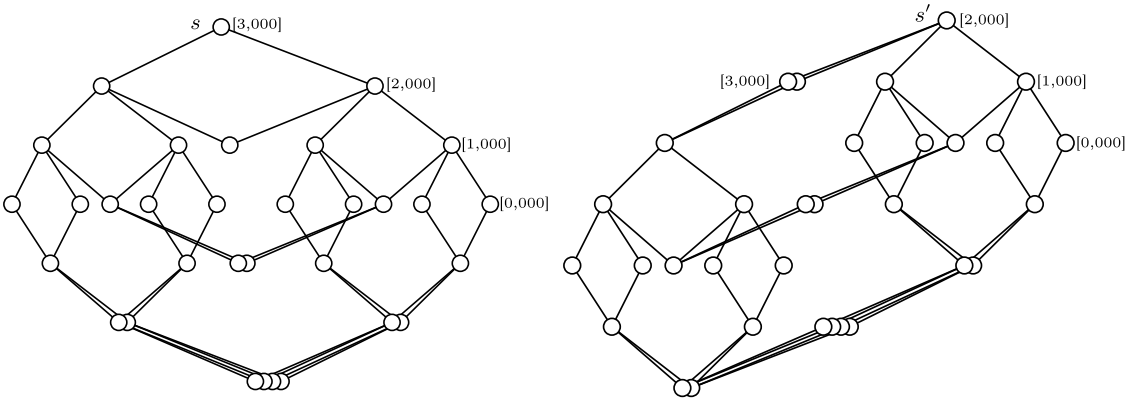


Fig. 2. *(left)* Embedding of a $BF(3)$ rooted at vertex $s = [3,000]$. *(right)* Embedding of a $BF(3)$ rooted at vertex $s' = [2,000]$.

This guarantees that, at the end of the embedding algorithm, all the vertices have different x-coordinates. As for the y coordinates, vertex $[d, 0^d]$ is placed at height d, see 18, while the other $2^d - 1$ vertices are placed at a height that corresponds to their distance from $[d, 0^d]$, see 21. At the end of the process, the root s will result in a vertex of degree two assigned with label $[d, 0^d]$. 2. left shows a $BF(3)$ graph rooted at vertex $s = [0,000]$, s has y-coordinate equal to three. Vertices of layer three are arranged between two butterflies of dimension two at the height that corresponds to their distance from s. Vertices of layer zero are arranged at level $y = 0$.

We recall that the described embedding uses a degree-two vertex s as root and arbitrarily assigns label $[d, 0^d]$ to it (note that all the degree-two vertices form orbit O_0 and hence they are pairwise equivalent). Assume now that an embedding with a degree-four vertex s' as root is needed, and that such a vertex belongs to O_i, with $0 < i \leq \lfloor d/2 \rfloor$. In this case, the embedding algorithm first selects s' as the vertex $[d - i, 0^d]$ and then changes the embedding realized by Hang(d) by updating the y-coordinate of each embedded vertex v as follows: $Y(v) = Y(s') - d(s', v)$. This yields the requested embedding of $BF(d)$ with s' as root. Fig. 2. right shows a $BF(3)$ rooted at the vertex $s' = [2,000]$.

Leader election algorithm. Let $C = (BF(d), \lambda)$ be any initial configuration with p-leaders. We describe here minString (cf. 2), the algorithm used to elect a leader among all the possible p-leaders of C. Let r be a p-leader of C and let v be the vertex where r is located. Basically, minString assigns a binary string to r so that the string encodes a visit of the entire configuration C starting from v. We remark that minString is executed concurrently by all robots on each possible p-leader. The p-leader that receives the smallest

Algorithm 2: minString(I, v, l).

```
1 Compute N_l(v), the set of vertices at distance l from v produced by Hang(d);
2 if N_l(v) = ∅ then
3     return '';
4 else
5     SL_min = min_{φ∈I} sl(φ, v, l);
6     I' = {φ ∈ I : sl(φ, v, l) = SL_min};
7     return (SL_min) + minString(I', v, l + 1);
```

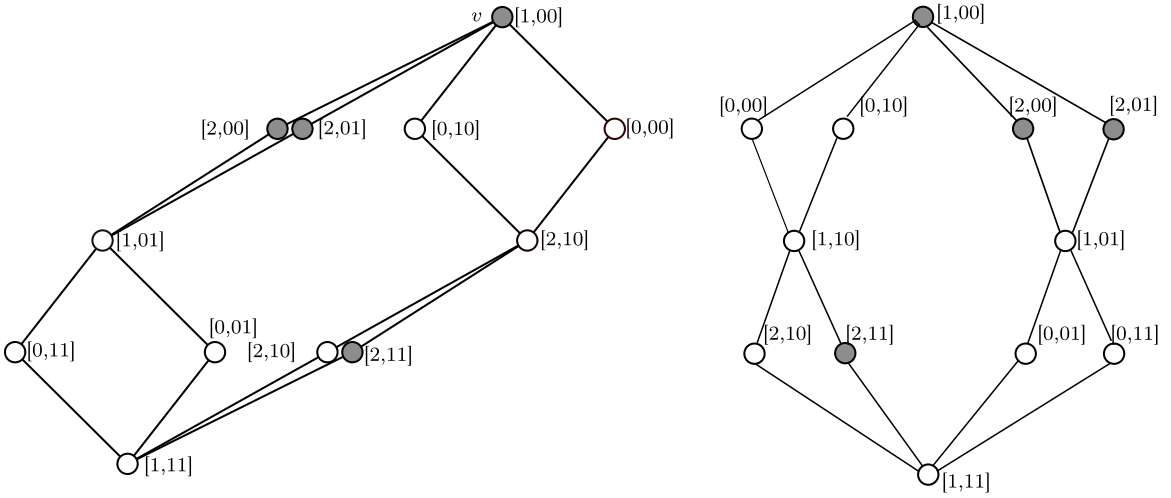


Fig. 3. *(left)* Embedding of a configuration defined on a $BF(2)$ and rooted at vertex $v = [1, 00]$. Grey vertices are those occupied by robots. *(right)* An equivalent representation of the same configuration showing its automorphisms at a glance.

string is elected as the leader. The visit performed by minString exploits the embedding described in the previous section: in particular, it assumes that a hanging of $BF(d)$ has been computed with a call to Hang(d). The binary string is built in blocks, each block is in correspondence to a level composed of the vertices having the same distance from v. Each value of the string, 1 or 0, corresponds to the presence or the absence, respectively, of a robot on each vertex of the level (read from left to right).

The string associated to r is computed by calling minString$(\text{Aut}(C), v, 0)$. Since r is a p-leader, each automorphism $\varphi \in \text{Aut}(C)$ fixes v, that is $\varphi(v) = v$. During the execution, at Lines 2 and 3, the algorithm checks if there are neighbors of v at level l if not, it returns the empty string. Otherwise, starting from 5, the algorithm calls the function sl for each automorphism $\varphi \in I$. The function sl works as follows. It takes as input an automorphism φ, the root v of the embedding produced by Hang(d), and a level l of the embedding, and returns the string associated with that level. It is called for each automorphism φ in I, where I is the set of automorphisms that make minimum the associated string up to level l. Hence, it finds the minimum string SL_{min} among the strings associated with level l which are returned by each call. At 6, the algorithm computes the subset I' of I given by all the automorphisms associated with the minimum string. Finally, the algorithm outputs the concatenation of the minimum string and the string obtained by a recursive call on the subset of automorphisms I' and the level $l + 1$.

For example, in Fig. 3 a p-leader is located on vertex v. The string associated with such a p-leader is computed by calling minString$(I, v, 0)$, where I contains all possible automorphisms. During the execution, $N_l(v) = \{v\}$, $SL_{\text{min}} = (1)$, $I' = I$ and the algorithm is called on I', v, and $l = 1$. $N_l(v)$ now contains four vertices. To visualize better the automorphisms see Fig. 3 right. The minimum string associated with level one is (0011) while now I' contains all the automorphisms that swap the vertices of the same levels but keep unchanged the minimum string associated with the vertices of level one. On level two, all the automorphisms in I' fix the vertices at that level, and the associated string is (00), while at level three the minimum string associated with that level is (0100). Indeed it is not possible to exchange the vertices of the graph to make the substring smaller at this level as the subset of the automorphisms prevents some exchanges between vertices. The string associated with level four is obviously (0). When the algorithm is called on level five, $N_l(v) = \emptyset$ it terminates returning the empty string. Hence, the minimum string associated to vertex v is (1)(0011)(00)(0100)(0).

Lemma 1. *Let $C = (BF(d), \lambda)$ be an initial configuration. If C is a leader configuration, the robots can elect a single leader in C.*

Proof. Assume first that there are orbits of $BF(d)$ containing only one p-leader. In such a case, robots agree that the requested leader corresponds to the p-leader which is the only robot on the orbit O_i, with minimum i, of the butterfly.

Assume now that there are no orbits of $BF(d)$ containing only one p-leader. Let O_i be the orbit of the butterfly with minimum i and containing p-leaders. Each robot executes $\mathsf{Hang}(d)$ and obtains an embedding of C on the Euclidean plane along with Cartesian coordinates with respect to a vertex v of O_i. Let r_1 and r_2 be two p-leaders respectively located on v_1 and v_2, where both the vertices are on O_i. We now prove that $s_1 = \mathsf{minString}(I_{v_1}, v_1, 0)$ and $s_2 = \mathsf{minString}(I_{v_2}, v_2, 0)$ are different strings. If we assume by contradiction that $s_1 = s_2$, then the embeddings of C rooted at the vertices v_1 and v_2 used by minString and produced by $\mathsf{Hang}(d)$ output the same minimum string, and these embeddings must be isomorphic. This would imply that r_1 and r_2 are equivalent, a contradiction to the hypothesis that r_1 and r_2 are p-leaders. In conclusion, this proves that it is possible to elect a leader among all the p-leaders located in O_i and, in turn, that is possible to elect a single leader in C. □ □

4. Gathering algorithm

In this section, we formalize an algorithm $\mathcal{A}$ that solves the gathering for synchronous robots in leader configurations defined on $BF(d)$. We assume $d \geq 2$ since the case $d = 1$ is trivial.

Definition 2. Let $C = (BF(d), \lambda)$, $d \geq 2$, be a configuration. The *leader vertex* v of C is defined as follows, only when the configuration satisfies one of the following cases:

- if C is a leader configuration, v corresponds to the vertex where the leader robot identified by 1 resides; we remark that v is selected as closer as possible to O_0.
- if C contains multiplicities, v is leader if O_i is the orbit of $BF(d)$ with the minimum i having a single occupied vertex, and v corresponds to such a vertex.

We first provide an overview of the strategy behind the provided algorithm. Given an initial configuration C admitting p-leaders, 1 is used to elect a leader robot r (assume located at O_k) and move all the robots so that (1) r moves to a lower orbit of $BF(d)$ (i.e. O_{k-1}) whereas the other robots leave O_{k-1}. This creates a new configuration C' where, by Observation 1, either r remains leader or r is located on a leader vertex (it depends on possible multiplicities created by robots different from r when they change the orbit and for which the final destination is chosen by the adversary). The same task is repeated until r reaches a boundary layer at O_0 (thus forming C''). Once the leader (vertex) is on O_0, the distance of all the robots from the leader is reduced. A special synchronous movement may be necessary when some robots must pass through O_0 (in particular, through the layer L complementary to the one in which the leader resides) to reach the leader. In such a case, the leader goes to O_1 while the other robots enter L. After that, r returns to O_0 (still preserving the property of occupying a leader vertex) while all the other continue reducing the distance from the leader. After this synchronization, all the robots can safely move toward the leader vertex along the shortest paths, until the gathering is accomplished.

To transform this strategy into an algorithm working under the LCM model, the gathering problem on $BF(d)$ must be subdivided into a set of very simple tasks where each task can be formalized according to three elements: (a) a precondition that can be evaluated by all robots according to perceived configuration, (b) a subset of robots selected for the movements, and (c) a specification of the move to be performed by each selected robot. The move should be performed by the robots selected in the task whose precondition is evaluated as true. In the remainder of this section, we formalize each task required by $\mathcal{A}$. Clearly, according to the properties holding in the initial configuration, not necessarily all the tasks must be performed. To enhance clarity, we also include an extended example that simulates one of the possible algorithm's execution. In general, the adversary decides the actual destination of robots when there are multiple possible destinations satisfying a move.

Task T_1. In this task, the leader robot is located in an internal layer. It moves so that its distance from a boundary layer is reduced and, at the same time, all the other robots move so that the leader remains unchanged.

- *Precondition* $\mathtt{pre}_1$*:* there exists a leader vertex (say v), v is located in O_k with $k > 0$, there is only one robot (say r) on v;
- *Selected robots:* all the elements of M_1, that is the set containing all robots in O_{k-1}, and the robot r;
- *Move* m_1*:* r moves to O_{k-1} and all robots in M_1 move to O_k.

Notice that at the end of the first execution of this task, r reduced its distance from a boundary layer. In case this distance is greater than zero, the precondition of this task remains true and hence the task is applied again until r reaches a boundary layer of O_0. When r reaches O_0, the subsequent task T_3 may start.

For example, let C_1 be an initial configuration shown in Fig. 4 left. This configuration is in Task T_1. The leader robot highlighted in bold is selected among the p-leaders (1) and it is located in the internal layer O_1. Robots apply the move m_1 starting from configuration C_1: the leader robot moves to layer O_0 while the robots located on O_0 move synchronously toward layer O_1, keeping the leader's identity unchanged. When the leader reaches O_0, the configuration C_3 is achieved, which is in T_3 as there is only the leader in O_0.

Task T_2. In this task, the leader vertex v is located in a boundary layer. Let r be the unique robot on v. In case O_0 contains other robots than r, all robots different from the leader (forming set M_2) will move to O_1.

- *Precondition* $\mathtt{pre}_2$*:* there exists a leader vertex (say v), there is only one robot (say r) on v, r resides in O_0, and O_0 contains two or more robots;
- *Selected robots:* all robots in M_2, i.e., all robots in O_0 different from r;
- *Move* m_2*:* all robots in M_2 move to O_1.

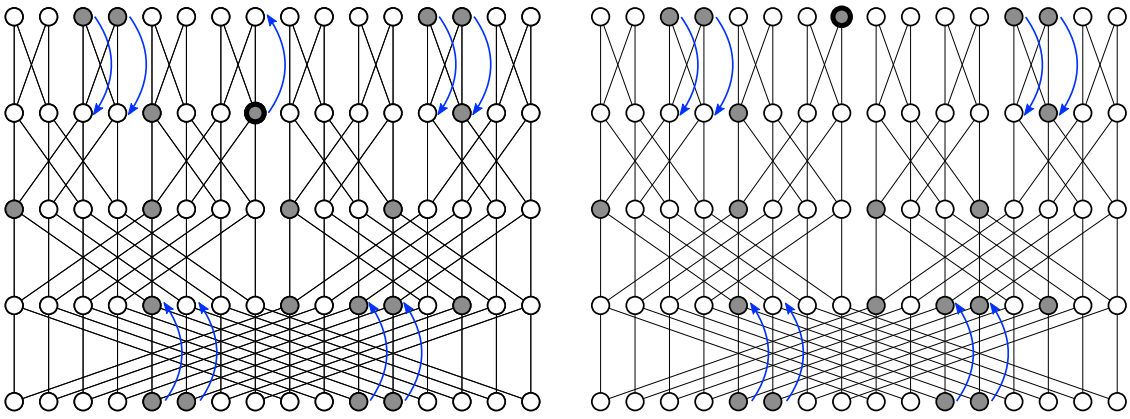


Fig. 4. (left) An initial configuration C_1 in task T_1; (right) An initial configuration C_2 in task T_2. Grey vertices represent robots. Arrows show the robots' movements and the target vertices. The leader robot is highlighted in bold.

At the end of this task, the leader vertex v results to be the only occupied vertex in O_0. This ensures that v remains the leader vertex (with just r on v) and task T_3 may start.

In Fig. 4 right we show another initial configuration, C_2, in which the leader robot is located in layer O_0 along with other robots. Here $\texttt{pre}_2$ holds and the configuration C_2 is in task T_2. All the robots on O_0 apart from the leader, apply the move m_2 and go from layer O_0 to O_1. After this move, the leader robot is the only one in O_0 and the configuration is in T_3.

Task T_3. In this task, all the robots that must pass through O_0 to reach the leader vertex v are involved (cf. Proposition 1). The goal is to minimize their distance from the leader vertex, but without letting them enter O_0 (this is to keep the leader unchanged). To formalize precondition and move, consider the following assumptions and definitions:

- there exists only one occupied vertex (say v) in O_0, and let r be the only robot on v,
- let L'' be the layer of O_1 furthest from v,
- let M_3 be the set of all the robots that must pass through O_0 to reach v.

According to such notation, in this task, the algorithm works as follows:

- *Precondition* $\texttt{pre}_3$*:* v is the leader vertex occupied only by robot r, and M_3 is not empty;
- *Selected robots:* all the robots in M_3;
- *Move* m_3*:* each robot $r' \in M_3$ moves to any vertex belonging to a r', r-shortest path not on O_0.

Note that this task may be repeated several times (consecutive rounds during the execution of $\mathcal{A}$) and when the last round is concluded, all the robots in M_3 will be located on L''.

After either task T_1 or T_2 have been completed, task T_3 begins. Fig. 5 left shows configuration C_3 where $\texttt{pre}_3$ is satisfied because the leader is the only robot in O_0 and there are robots located on the $BF(d-1)$ sub-butterfly in which the leader does not reside. These robots belong to the M_3 set. To achieve gathering, robots in M_3, must reach the sub-butterfly where the leader is located. To do so, they have to travel through layer O_0. During task T_3, each robot in M_3 applies move m_3 and goes toward the vertices of layer O_1 furthest from the leader. Two rounds are required to reach it, in this example. When the configuration in Fig. 5 right is reached, $\texttt{pre}_4$ becomes true and the configuration obtained, C_4 in task T_4.

Task T_4. This task starts immediately after the conclusion of T_3 and operates on the same set of robots. The goal is to further reduce the distance of these robots from the leader vertex v, but now a special move is needed to keep always the same leader. Consider the following assumptions and definitions:

- there exists only one occupied vertex (say v) in O_0,
- let L' (L'', resp.) be the layer of O_1 closest to (furthest from, resp.) v,
- let L''' be the layer complementary to the one in which v resides.

According to such notation, in this task, the algorithm works as follows:

- *Precondition* $\texttt{pre}_4$*:* v is the leader vertex, r is the only robot on v, all robots that must pass through L''' to reach v are on L'', and L'' is occupied.
- *Selected robots:* r and all robots in O_1.
- *Move* m_4*:* r moves to O_1, each robot on L' moves to O_2, each robot on L'' moves to L'''.

During this task, see Fig. 5 right, there are some robots located on O_1 on the $BF(d-1)$ where there is no leader. This time, several robots must perform a coordinated movement to keep the leader's identity unchanged. The leader moves toward layer O_1, the robots

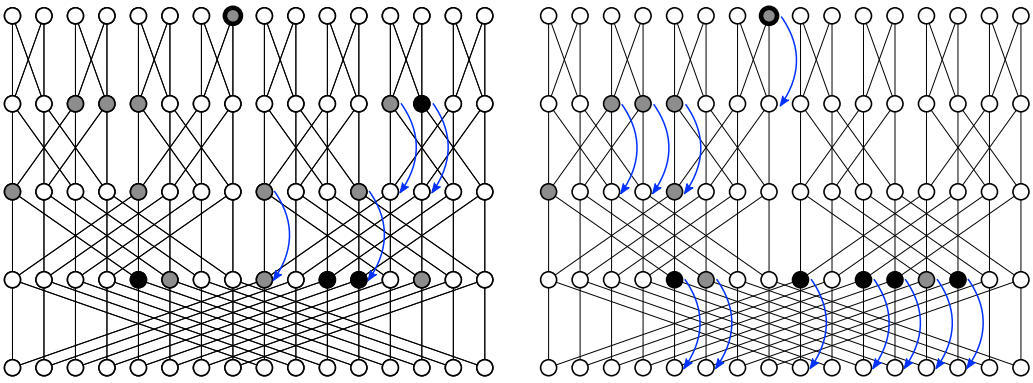


Fig. 5. (left) A configuration C_3 in task T_3; (right) A configuration C_4 in task T_4. Black vertices represent multiplicities.

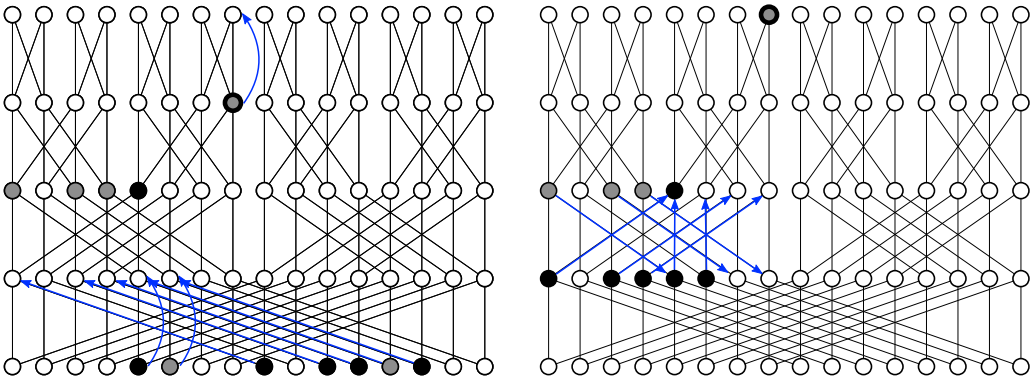


Fig. 6. (left) A configuration C_5 in task T_5; (right) A configuration C_6 in task T_6.

lying on the layer O_1 closer to the leader move to O_2, and the robots on O_1 furthest from the leader move to O_0. At the end of the move, two possible cases must be analyzed:

- There are two or more occupied vertices in O_0. Then, the leader is on layer O_1, layer O_0 closest to the leader has no robots, no robots must pass through O_0 to reach the leader and there are robots on layer O_0 furthest from the leader. Hence, precondition $\texttt{pre}_5$ becomes true.
- There is only one occupied vertex (say v') in O_0. By definition, vertex v' becomes the leader. Since all robots reside on the same sub-butterfly $BF(d-1)$ which v' belongs to, precondition $\texttt{pre}_6$ becomes true.

Task T_5. In this task, the leader robot moves on a vertex in O_0, and robots on O_0 move on vertices in O_1. Consider the following assumptions and definitions:

- there exists only one occupied vertex (say v) in O_1,
- let L' (L'', resp.) be the layer of O_0 closest to (furthest from, resp.) v,
- let M_5 be the set containing all robots on L''.

According to such notation, in this task, the algorithm works as follows:

- *Precondition* $\texttt{pre}_5$*:* v is a leader vertex, there is only one robot (say r) on v, there are no robots on L', there are no robots that must pass through O_0 to reach v, and M_5 is not empty;
- *Selected robots:* r and all robots in M_5;
- *Move* m_5*:* r moves from O_1 to O_0, each robot $r' \in M_5$ moves toward a vertex in O_1 belonging to a r', r-shortest path.

When this task finishes, the leader v is the unique occupied vertex on a boundary layer, r is the unique robot on v, and all the remaining robots are at a distance at most $d-1$ from r.

Fig. 6 left shows configuration C_5 which is T_5. During this task, the leader robot moves back to the closest boundary layer O_0 while the robots located on layer O_0 and furthest from the leader, move toward a vertex in O_1 that reduces their distance from the leader. After this movement, the leader is the unique robot occupying a vertex of a boundary layer, and all the remaining robots are at a distance at most $d-1$ from the leader. At this stage, $\texttt{pre}_6$ holds as all the robots reside in the same sub-butterfly $BF(d-1)$, and the leader's identity is preserved. Configuration C_6 is reached and task T_6 can start, see Fig. 6 right.

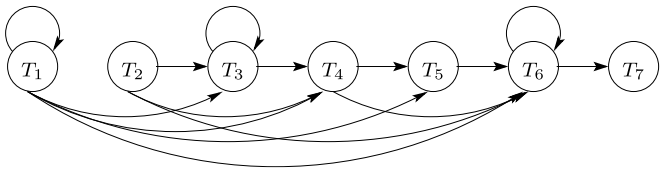


Fig. 7. Transition graph generated by $\mathcal{A}$.

Task T_6. The algorithm $\mathcal{A}$ finalizes the gathering by moving all robots on the leader vertex v. Notice that, after the first round in which this task is executed, v may contain a multiplicity. $\mathcal{A}$ works as follows:

- *Precondition* $\texttt{pre}_6$*:* there exists a leader vertex (say v) in O_0 and all robots reside on the same sub-butterfly $BF(d-1)$ which v belongs to;
- *Selected robots:* all the elements of M_6, i.e., the set containing all robots not located on v;
- *Move* m_6*:* each robot $r' \in M_6$ selects a vertex along a shortest path to v as destination.

During task T_6, from Fig. 6 right, all robots not located on the leader vertex, move toward it. This task is repeated for several rounds until the gathering is accomplished and $\texttt{pre}_7$ holds. Fig. 6 right shows the robots' movements during the first round.

Task T_7. It represents the case in which all robots recognize that the gathering is accomplished and hence none of them moves.

- *Precondition* $\texttt{pre}_7$*:* all the robots reside on a same vertex v, i.e., the configuration is final;
- *Selected robots:* none.

4.1. Correctness and complexity

The algorithm $\mathcal{A}$ has been designed according to the general methodology proposed in [12]. To detect which task must be accomplished in any configuration observed during an execution, a **predicate** P_i is assigned to task T_i, for each i. P_i is defined as follows: $P_i = \texttt{pre}_i \wedge \neg(\texttt{pre}_{i+1} \vee \texttt{pre}_{i+2} \vee \ldots \vee \texttt{pre}_7)$.

The predicate is evaluated in the `Compute` phase, based on the view acquired during the `Look` phase. As soon as the robots recognize that a task T_i must be accomplished, move m_i associated with that task is performed by a subset of designed robots. In the `Compute` phase, each robot evaluates – for the perceived configuration and the provided input – the predicates starting from P_7 and proceeding in the reverse order until a true precondition is found. In case all predicates $P_7, \ldots, P_3$ are evaluated false, then either T_2 or T_1 is performed. Indeed, one of the two preconditions certainly holds because the algorithm is working on a configuration with p-leaders. Summarizing, the provided algorithm can be used by each robot r in the `Compute` phase as follows:

– *if r detects that P_i holds, then it applies move m_i associated with T_i.*

Consider now an execution of $\mathcal{A}$, and assume that a task T_i, $i < 7$, is performed for the current configuration C. If $\mathcal{A}$ transforms C into C' and this new configuration has to be assigned the task T_j, then we say that $\mathcal{A}$ can generate a transition from T_i to T_j. The set of all possible transitions of $\mathcal{A}$ determines a directed graph called *transition graph* (cf. Fig. 7). Of course, in the transition graph, there should be a sink vertex corresponding to the case in which robots simply recognize that the problem is solved and no move is performed.

According to the proposed methodology, in [12] it is shown that the correctness of $\mathcal{A}$ can be obtained by proving that all the following properties hold:

- H_1: for each task T_i, the tasks reachable from T_i through transitions are exactly those represented in the transition graph (i.e., the transition graph is correct);
- H_2: possible cycles in the transition graph (including self-loops) must be performed a finite number of times;
- H_3: unsolvable configurations are not generated by $\mathcal{A}$ (concerning the addressed problem, this means that $\mathcal{A}$ does not generate configurations without a leader vertex).

Lemma 2. *Let $C = (BF(d), \lambda)$, $d \geq 2$, be any configuration of $n \geq 2$ synchronous robots with p-leaders. During any execution of $\mathcal{A}$ on C, properties H_1, H_2 (limited to self-loops), and H_3 hold in each task T_i, $1 \leq i < 7$.*

Proof. In T_1, there exists a leader vertex v located in O_k, with $k > 0$, with only one robot r located on v. All the robots in O_{k-1} move to O_k, while r moves to O_{k-1}. Note that, after the move, r will be the only robot in O_{k-1} (this implies that H_3 holds). The move generates transitions to T_1 (when $k > 1$) or $T_3, \ldots, T_6$ (when $k = 1$). In case $k = 1$, task T_2 cannot be reached since r would be the only robot in O_0. Notice that transitions to T_1 can be generated at most $\lfloor \frac{d}{2} \rfloor - 1$ times. Summarizing, H_1, H_2, and H_3 hold in T_1.

In T_2, a leader vertex v exists in O_0 with only one robot r located on it, and O_0 contains more than one robot. Then, all robots in O_0 different from r move to O_1. Clearly, there are no transitions to the same task (hence no self-loop in the transition graph), while transitions to T_3, T_4, and T_6 can be generated (whereas the precondition of T_5 cannot hold). Notice that v remains the leader vertex for each possible transition. Hence, H_1, H_2, and H_3 hold in T_2.

In T_3, there is a leader vertex v in O_0, and r is the unique robot on v. All robots that must pass through O_0 to reach v and are not located on L'' (the layer in O_1 furthest from v) move decreasing their distance from L''. Notice that the leader vertex remains unchanged (H_3 holds). The precondition holds until all the robots selected for moving reach L''. Hence, there can be transitions to the same task for at most $d-1$ times since the furthest robot from v can be at distance at most $2d$ initially (hence H_2 holds). The only further possible transition is the one towards T_4. Summarizing, H_1, H_2, and H_3 hold in T_3.

In T_4, there is a leader vertex v in O_0, and r is the unique robot on v. All robots that must pass through O_0 to reach v are now located on L'' (the layer in O_1 furthest from v). Concerning movements, r moves to the layer L' of O_1 closest to r, each robot on L' moves to O_2, the next orbit, and each robot on L'' moves to L''' (i.e., the boundary layer complementary to the one in which v resides). Let C' be any created configuration. In C', r occupies a leader vertex (H_3 holds). If in L''' there is just one vertex v' occupied because all the robots in L'' moved toward the same vertex v', then v' is a leader vertex as well, and the precondition of T_6 holds. If, instead, in C' there are more vertices occupied in L''', then the precondition of T_5 holds. Hence, H_1, H_2, and H_3 hold in T_4.

In T_5, there is a leader vertex v in O_1, and r is the unique robot on v. Denoting as L' and L'' the layers of O_0 closest to and furthest from v, respectively, the precondition of the task assures that L' is unoccupied whereas there are robots on L''. Moreover, there are no robots that must enter O_0 to reach v. In such a situation, r moves from O_1 to O_0, and each robot on L'' moves to O_1 along a shortest path toward v, thus maintaining the original distance from r. It is clear that, in any created configuration, r is located in a leader vertex (hence, H_3 holds), and all robots are now within the same sub-butterfly $BF(d-1)$ (hence the precondition of T_6 holds and only this transition is possible). This concludes the proof for this task.

In T_6, there exists a leader vertex v in O_0 and all robots reside on the same sub-butterfly $BF(d-1)$. Since all robots move toward v through shortest paths, from Observation 1 it follows that in each generated configuration the same precondition remains valid until all robots reach v. Hence, from T_6, only transitions to T_6 and T_7 can be generated. Transitions to T_6 can be generated at most $d-2$ times since the distance from the moving robots and v reduces in each transition. Finally, as observed, the precondition remains true as long as there are robots not on v, hence v remains a leader vertex. This implies that H_1, H_2, and H_3 hold in T_6. □ □

Theorem 1. *If $C = (BF(d), \lambda)$, $d \geq 2$ is any initial configuration of $n \geq 2$ synchronous robots with p-leaders, then $\mathcal{A}$ optimally solves the gathering problem on C in $\Theta(d)$ round.*

Proof. When C is provided as input to $\mathcal{A}$, according to the definition of predicates P_i we are sure that exactly one task is assigned to robots and hence executed. 2 ensures that properties H_1, H_2 and H_3 hold for each task $T_1, \ldots, T_6$. This implies that $\mathcal{A}$ can proceed in its execution by following the transition graph reported in 7. Since there are only forward transitions (i.e., $j > i$ for each transition $T_i \rightarrow T_j$), and since each self-loop is repeated a finite number of times (cf proof of 2), we are sure that in a finite number of rounds, a configuration C' assigned to task T_7 is generated. Formally, for each execution $\mathbb{E} : C = C(0), C(1), C(2), \ldots$ of $\mathcal{A}$, there exists a finite time $t > 0$ such that in $C(t)$ all the robots are located in the same vertex (i.e., the gathering problem is solved).

Concerning the time (i.e., the number of rounds) required by $\mathcal{A}$, from the transition graph, it can be observed that there are no cycles but tasks with self-loops. Since each of such loops can be repeated less than d times (cf proof of 2), the total time is trivially bounded by the number of tasks times d. This leads to a final $\Theta(d)$ time complexity since d is the trivial lower bound required by any algorithm when in the initial configuration there are two robots at maximum distance $2d$. □ □

5. About non-leader and non-partitive configurations

In the previous section, we have shown that $\mathcal{A}$ can solve the gathering problem in each configuration with p-leaders. Moreover, in [23, Theorem 2] it is shown that any partitive configuration is ungatherable. In this section, we address the case of any configuration C which is, at the same time, *non-leader* (i.e., without p-leader robots) and *non-partitive*. To this end, we first provide a list of technical results ending with the 2 (which characterizes non-leader and non-partitive configurations in terms of properties about orbits), and then we discuss possible difficulties in determining if the gathering problem can be solved in these configurations.

Lemma 3. *Let $C = (BF(d), \lambda)$ be an initial configuration. If C contains an odd number of robots, then an occupied vertex v can be elected as leader.*

Proof. Select the orbit O_i of $BF(d)$ with an odd number of robots and minimum index i. Let ℓ be the layer in O_i with an odd number of robots and let R' be the set of these robots.

Assume first that $i \neq d/2$. Identify in O_0 the boundary layer L' closer to ℓ (L'' farther from ℓ, resp.). If $\ell \neq L'$, consider the subgraph induced by the vertices in $BF(d)$ without L'. If $\ell = L'$, consider the subgraph induced by the vertices in $BF(d)$ without L''. In both cases, the subgraphs are composed of two connected components, that are two butterflies graphs $BF(d-1)$. Select the $BF(d-1)$ with an odd number of robots in R'. Apply recursively the procedure on this $BF(d-1)$, where ℓ is always the layer with robots in R'. The procedure stops when a $BF(0)$ with a single occupied vertex v is obtained.

If $i = d/2$, consider the subgraph induced by the vertices in $BF(d)$ without O_0. According to the selection of orbit O_i, all the other orbits have an even number of robots, therefore also O_0. The subgraph obtained by removing O_0 is composed of four connected components, which are four butterfly graphs $BF(d-2)$. Then, there are either one or three $BF(d-2)$ with an odd number of robots. In the first case, the single $BF(d-2)$ with an odd number of robots on ℓ is chosen, in the latter case, it is selected the $BF(d-2)$ lying between the other two. The procedure is repeated until a $BF(0)$ with its unique occupied vertex v is obtained. □ □

This lemma can be reformulated as follows.

Corollary 1. *Let $C = (BF(d), \lambda)$ be an initial configuration. If C is non-leader, then it contains an even number of robots.*

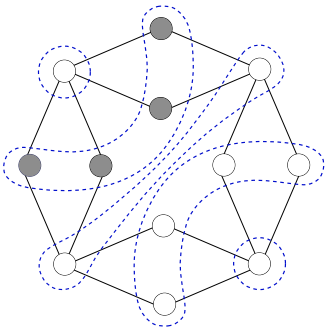


Fig. 8. An example of a non-partitive and non-leader configuration. Gray vertices represent robots and dashed closed curves highlight orbits. There are two ep-leaders.

We have already remarked that each orbit O_i of $BF(d)$ is composed of the complementary layers i and $d-i$ for each $0 \leq i \leq \lfloor d/2 \rfloor$, with the exception of the intermediate orbit $O_{d/2}$ that is composed of just one layer when d is even. Hence, $|O_i| = 2^{d+1}$ for each $0 \leq i < \lfloor d/2 \rfloor$, $|O_{d/2}| = 2^{d+1}$ if d is odd, and $|O_{d/2}| = 2^d$ if d is even. The following result extends this analysis to orbits of configurations defined on butterfly networks.

Lemma 4. *Let $C = (BF(d), \lambda)$ be a configuration and Q an orbit of C. Then, $|Q| = 2^k$ for some $k \geq 0$.*

Proof. Let $H = \text{Aut}(BF(d))$ and $K = \text{Aut}(C)$. Since all the possible automorphisms in K are in H then K is a subgroup of H. By the Lagrange's theorem $|K|$ divides $|H|$. By the Orbit-Stabilizer theorem $|Q|$ divides $|K|$ (see [5] for both theorems). Then $|Q|$ divides $|H|$. Since the automorphism group of $BF(d)$ has size $2^{2^{d+1}-1}$ for $d > 0$, that is $|K| = 2^{2^{d+1}-1}$, the statement follows. For $d = 0$, $BF(d)$ has only one vertex and the statement holds choosing $k = 0$. □ □

Lemma 5. *Let $C = (BF(d), \lambda)$ be an initial configuration. If $|Q| > 1$ for each orbit Q of C, then C is partitive.*

Proof. Since the smallest orbit of C contains more than one element, 4 implies $|Q| = 2^k$, $k \geq 1$, for each orbit Q of C. From this property, it can be easily derived that C is partitive since each orbit can be partitioned into subsets containing two elements. □ □

Corollary 2. *Let $C = (BF(d), \lambda)$ be an initial configuration. If C is non-leader and non-partitive, then (1) there are orbits of C with just one element, and (2) all orbits of C with one element are unoccupied.*

Proof. Since C is non-partitive, 5 implies that in C there exists an orbit Q such that $|Q| = 1$. Since C is non-leader, all orbits Q such that $|Q| = 1$ are unoccupied. □ □

The following definition is introduced to simplify the terminology about non-leader and non-partitive configurations.

Definition 3. Let $C = (BF(d), \lambda)$ be any initial configuration. A vertex v of $BF(d)$ is called *empty potential-leader* (ep-leader, for short) if $\lambda(v) = 0$ and $\varphi(v) = v$ for each $\varphi \in \text{Aut}(C)$.

This definition, along with 2 and 2, allow us to say that *each non-leader and non-partitive configuration C contains ep-leaders but not p-leaders*. In other words, in C there must be orbits consisting of only one vertex and such orbits are all unoccupied.

Fig. 8 shows a very simple configuration C that is both non-partitive and non-leader (indeed, it contains two ep-leaders and no p-leaders). It is easy to observe that the Gathering problem can be trivially solved in C since just a single move leads all robots to finalize the gathering over the closer ep-leader. From this example, it can be deduced that there are non-leader and non-partitive configurations for which the Gathering problem can be solved. In general, however, it is difficult to state and prove that the problem can be solved in *all* configurations of this kind. One way to see this is as follows. In [23, Theorem 3] it is proved that if a configuration defined on an arbitrary graph $G = (V, E)$ is partitive on the subgraph induced by $V' \subseteq V$, then the gathering, whenever realizable, must necessarily be finalized on a vertex belonging to $V \setminus V'$. In the case under discussion, if C is a non-leader and non-partitive configuration, then the gathering, whenever realizable, must be finalized on an ep-leader v.

This implies the following properties/observations holding during the entire execution of any possible gathering algorithm $\mathcal{A}'$ for C:

- $\mathcal{A}'$ should be provided with a leader election phase that allows robots to agree on a specific ep-leader to be used as final destination vertex. A strategy could be to select the ep-leader whose sum of distances from all the robots is minimum and, in case of a tie, the ep-leader with minimum string as defined in 3. According to this strategy, assume in what follows that v is the elected ep-leader;
- we first recall that the strategy underlying the algorithm $\mathcal{A}$ provided in 4 is based on carefully moving the elected p-leader along the layers of the butterfly to avoid the formation of partitive configurations during the execution. For $\mathcal{A}'$, this is not possible since the elected target v is unoccupied;
- according to the previous remark, v should remain the selected ep-leader until the last move of $\mathcal{A}'$ in which the gathering is finalized. Fig. 9.left shows an example in which this is possible. Robots in orbit O_1 are at distance one from v, hence they will move only in the last execution round of $\mathcal{A}'$. The robots in orbit O_2 will be moved initially, needing only two rounds to reach

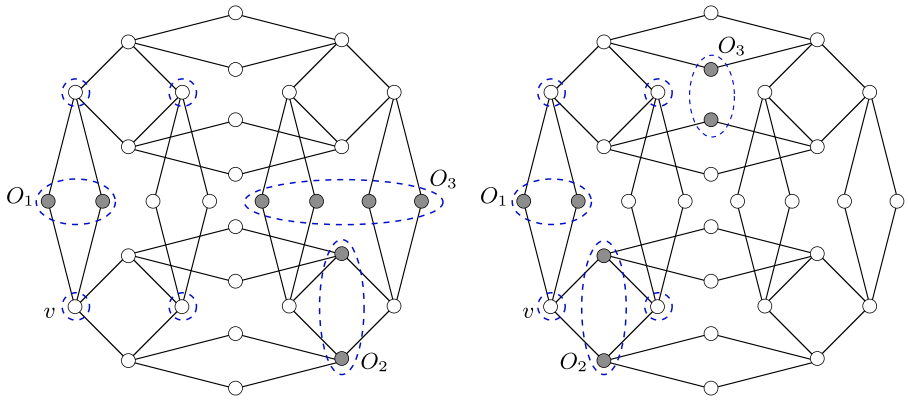


Fig. 9. Additional examples of non-partitive and non-leader configurations. Dashed ovals are used to highlight some orbits.

distance one from v. Then the four robots in O_3 will reach a vertex at distance one from v by moving along a shortest path having length four. Note that multiplicities will be created during this phase. Finally, by a single additional move, all robots will gather on v.

- Fig. 9.right shows an example in which this strategy fails. There, orbits O_1 and O_2 contain robots which are already at distance one from v. When robots in O_3 move along any shortest path toward v, a partitive configuration is created, and hence the gathering cannot be finalized any longer.

  As a possible variation in the proposed strategy, robots in O_3 could reach v through an alternative path (the crossing of which guarantees the formation of non-partitive configurations).

Furthermore, by assuming robots endowed with the *strong* multiplicity detection, in the last case the gathering could be also finalized according to a different approach. An ep-leader v could be initially elected as in the above approach, then robots in the two closer orbits can move toward v (thus creating a multiplicity in v made up of a number of robots greater than the number of robots not on v), and finally, all the robots not on v can agree on finalizing the gathering on that special multiplicity. It is worth noting that this strategy may fail when the initial multiplicity in v contains a small number of robots. In fact, in such a situation, when moving robots not on v, different but equivalent multiplicities can be created, and this could lead to a partitive configuration.

As a final observation, note that even if it were possible to determine an algorithm $\mathcal{A}'$ capable of solving the gathering problem on all the non-partitive and non-leaders configurations, for a complete characterization of the problem on all the butterflies, a final step would be necessary: $\mathcal{A}$ and $\mathcal{A}'$ should be defined to be "composable" into a single algorithm $\mathcal{A}^*$. If so, $\mathcal{A}^*$ would be able to solve the gathering problem in any initial configuration C that is non-partitive.

6. Concluding remarks

We considered the Gathering problem in the $\mathcal{OBLOT}$ model for synchronized robots moving on Butterfly graphs. We provided a Gathering algorithm for all the configurations in which it is possible to elect a robot as a leader (leader configurations). The difficulties provided by the isometries of the Butterfly topology have been addressed by suitably choosing an embedding of the Butterfly on the Euclidean plane. To fully characterize the problem on Butterfly networks, it is important to provide a Gathering algorithm also for all the configurations that are potentially gatherable, therefore not partitive, but not admitting a leader. These are the configurations in which leader vertices are not occupied by robots (ep-leaders). We have extensively discussed the difficulties of designing a unifying strategy to gather all these configurations. Even if it were possible to determine an algorithm capable of solving the gathering problem for these configurations, it would be necessary to compose this algorithm with the algorithm solving the leader configurations. In case of success, the resulting algorithm would be able to solve the gathering problem in any initial configuration that is non-partitive, thereby providing a complete characterization of the problem. As a future work, it would also be interesting to study the Gathering problem on different topologies, like (not oriented) Hypercubes or Cube Connected Cycle graphs.

CRediT authorship contribution statement

Serafino Cicerone: Writing - review & editing, Writing - original draft, Visualization, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization; **Alessia Di Fonso:** Writing - review & editing, Writing - original draft, Visualization, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization; **Gabriele Di Stefano:** Writing - review & editing, Writing - original draft, Visualization, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization; **Alfredo Navarra:** Writing - review & editing, Writing - original draft, Visualization, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization.

Data availability

No data was used for the research described in the article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The work has been partially supported by the European Union - NextGenerationEU under the Italian Ministry of University and Research (MUR) National Innovation Ecosystem grant ECS00000041 - VITALITY - CUP J97G22000170005, by the Italian Ministry of Economic development (MISE) under the project "SICURA - Casa intelligente delle tecnologie per la sicurezza", CUP C19C200005200004, and by the Italian National Group for Scientific Computation (GNCS-INdAM).

References

[1] N. Agmon, D. Peleg, Fault-tolerant gathering algorithms for autonomous mobile robots, SIAM J. Comput. 36 (1) (2006) 56–82.
[2] F. Bonnet, M. Potop-Butucaru, S. Tixeuil, Asynchronous gathering in rings with 4 robots, 9724 of 5th Int.'l Conf. on Ad-hoc, Mobile, and Wireless Networks (ADHOC-NOW), Springer, 2016.
[3] K. Bose, M.K. Kundu, R. Adhikary, B. Sau, Optimal gathering by asynchronous oblivious robots in hypercubes, in: Wireless Networks and Distributed Robotics (Algosensors), volume 11410 of LNCS, 2019, pp. 102–117. Proc. 20th Int.'l Symp. on Algorithms and Experiments for Sensor Systems.
[4] Q. Bramas, A. Lamani, S. Tixeuil, Stand up indulgent gathering, Theor. Comput. Sci. 939 (2023) 63–77.
[5] N. Carter, Visual Group Theory, American Mathematical Society, 2009.
[6] S. Cicerone, A.D. Fonso, G.D. Stefano, A. Navarra, Arbitrary pattern formation on infinite regular tessellation graphs, Theor. Comput. Sci. 942 (2023) 1–20.
[7] S. Cicerone, G.D. Stefano, A. Navarra, Asynchronous arbitrary pattern formation: the effects of a rigorous approach, Distr. Comput. 32 (2) (2019) 91–132.
[8] S. Cicerone, G.D. Stefano, A. Navarra, Embedded pattern formation by asynchronous robots without chirality, Distr. Comput. 32 (4) (2019) 291–315.
[9] S. Cicerone, G.D. Stefano, A. Navarra, On gathering of semi-synchronous robots in graphs, in: Proc. 21st Int,'l Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS), volume 11914 of LNCS, 21st Int,'l Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS), volume 11914 of LNCS, Springer, 2019, pp. 84–98.
[10] S. Cicerone, G.D. Stefano, A. Navarra, Gathering robots in graphs: the central role of synchronicity, Theor. Comput. Sci. 849 (2021) 99–120.
[11] S. Cicerone, G.D. Stefano, A. Navarra, Solving the pattern formation by mobile robots with chirality, IEEE Access 9 (2021) 88177–88204.
[12] S. Cicerone, G.D. Stefano, A. Navarra, A structured methodology for designing distributed algorithms for mobile entities, Inf. Sci. 574 (2021) 111–132.
[13] M. Cieliebak, P. Flocchini, G. Prencipe, N. Santoro, Distributed computing by mobile robots: gathering, SIAM J. Comput. 41 (4) (2012) 829–879.
[14] J. Czyzowicz, L. Gasieniec, A. Pelc, Gathering few fat mobile robots in the plane, Theor. Comput. Sci. 410 (6–7) (2009) 481–499.
[15] D. Gianlorenzo, G.D. Angelo, R. Stefano, A. Klasing, Navarra, Gathering of robots on anonymous grids and trees without multiplicity detection, Theor. Comput. Sci. 610 (2016) 158–168.
[16] D. Gianlorenzo, G.D. Angelo, A. Stefano, Navarra, Gathering asynchronous and oblivious robots on basic graph topologies under the look-compute-move model, in: Search Theory: A Game Theoretic Perspective, Springer, 2013, pp. 197–222.
[17] D. Gianlorenzo, G.D. Angelo, A. Stefano, Navarra, Gathering on rings under the look-compute-move model, Distrib. Comput. 27 (4) (2014) 255–285.
[18] D. Gianlorenzo, G.D. Angelo, A. Stefano, Navarra, Gathering six oblivious robots on anonymous symmetric rings, J. Discrete Algorithms 26 (2014) 16–27.
[19] D. Gianlorenzo, G.D. Angelo, A. Stefano, N. Navarra, K. Nisse, Suchan, Computing on rings by oblivious robots: a unified approach for different tasks, Algorithmica 72 (4) (2015) 1055–1096.
[20] D. Gianlorenzo, A. Angelo, N. Navarra, Nisse, A unified approach for gathering and exclusive searching on rings under weak assumptions, Distrib. Comput. 30 (1) (2017) 17–48.
[21] D. Mattia, G.D. Emidio, D. Stefano, A. Frigioni, Navarra, Characterizing the computational power of mobile robots on graphs and implications for the euclidean plane, Inf. Comput. 263 (2018) 57–74.
[22] G. Di, Stefano, A. Navarra, Gathering of oblivious robots on infinite grids with minimum traveled distance, Inf. Comput. 254 (2017) 377–391.
[23] G. Di, Stefano, A. Navarra, Optimal gathering of oblivious robots in anonymous graphs and its application on trees and rings, Distrib. Comput. 30 (2) (2017) 75–86.
[24] Y. Dieudonné, A. Pelc, D. Peleg, Gathering despite mischief, ACM Trans. Algorithms 11 (1) (2014).
[25] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Asynchronous gathering in a dangerous ring, Algorithms 16 (5) (2023) 222.
[26] P. Flocchini, G. Prencipe, N. Santoro (Eds.), Lecture Notes in Computer Science, 11340, Springer, 2019.
[27] P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, Gathering of asynchronous robots with limited visibility, Theor. Comput. Sci. 337 (2005) 147–168.
[28] G.P. Paola Flocchini, N. Santoro (Eds.), Distributed Computing by Oblivious MobileRobots. sSynthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers, 2012.
[29] S. Guilbault, A. Pelc, Gathering asynchronous oblivious agents with local vision in regular bipartite graphs, Theor. Comput. Sci. 509 (2013) 86–96.
[30] T. Izumi, S. Souissi, Y. Katayama, N. Inuzuka, X. Défago, K. Wada, M. Yamashita, The gathering problem for two oblivious robots with unreliable compasses, SIAM J. Comput. 41 (1) (2012) 26–46.
[31] T. Izumi, T. Izumi, S. Kamei, F. Ooshita, Time-optimal Gathering Algorithm of Mobile Robots with Local Weak Multiplicity Detection in Rings, IEICE Trans. Fundam. Electron. Commun. Comput. Sci., 96-A (6) (2013) 1072–1080.
[32] S. Kamei, A. Lamani, F. Ooshita, S. Tixeuil, K. Wada, Asynchronous Gathering in a Torus, Technical Report, 17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
[33] R. Klasing, A. Kosowski, A. Navarra, Taking advantage of symmetries: gathering of many asynchronous oblivious robots on a ring, Theor. Comput. Sci. 411 (2010) 3235–3246.
[34] R. Klasing, E. Markou, A. Pelc, Gathering asynchronous oblivious mobile robots in a ring, Theor. Comput. Sci. 390 (2008) 27–39.
[35] G. Antonio, D. Luna, P. Flocchini, L. Pagli, G. Prencipe, N. Santoro, G. Viglietta, Gathering in dynamic rings, Theor. Comput. Sci. 811 (2020) 79–98.
[36] P.D. Manuel, I. Mostafa, I. Abd-El-Barr, B. Rajasingh, Rajan, An efficient representation of benes networks and its applications, J. Discrete Algorithms 6 (1) (2008) 11–19. Ok.
[37] A. Pelc, Deterministic gathering with crash faults, Networks 72 (2) (2018) 182–199.
[38] I. Suzuki, M. Yamashita, Distributed anonymous mobile robots: formation of geometric patterns, SIAM J. Comput. 28 (4) (1999) 1347–1363.