Breaking Symmetries on Tessellation Graphs via Asynchronous Robots*

Serafino Cicerone

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università degli Studi dell'Aquila, Italy. serafino.cicerone@univaq.it

Abstract. We consider the coordination of autonomous mobile robots operating in the standard Look–Compute–Move cycles. Robots are assumed to be very weak computational units, since they are asynchronous, oblivious, anonymous, silent and execute the same distributed algorithm. In this area, the main focus has been on the important class of *Pattern Formation* problems, where the robots are required to arrange themselves to form a given geometric shape. This class of problems has been extensively studied in the Euclidean plane, whereas few results exist when robots move on a discretization of the plane, like infinite grids. In infinite grids, in order to form any pattern, the problem of breaking symmetries clearly emerges. Breaking the symmetry by moving some leader robot is not a straightforward task due to the movement restrictions as all the adjacent nodes of the leader may be occupied. Due to the asynchrony of robots, this fact greatly increases the difficulty of the problem. We assume regular tessellation graphs as discretization of the Euclidean plane, and we devise an algorithm able to solve the *Symmetry Breaking* problem on both the square and triangular grids. The algorithm is proposed so that it can be also combined with other modules.

1 Introduction

The coordination of autonomous mobile robots has long been object of study in several fields, including robotics, control, AI, as well as distributed computing. Within distributed computing, in particular, extensive research efforts have been conducted in the last two decades to investigate the computational and complexity issues arising in distributed systems composed of a team of mobile computational entities moving and operating in a Euclidean space (e.g., see [24]).

These entities, called robots, are *autonomous* (no centralized control), *anonymous* (they are identical in their external appearance, no unique identifiers), *homogeneous* (have the same capabilities and execute the same algorithm), *silent* (they have no explicit means of direct communication), and *disoriented* (no common coordinate system, no common left-right orientation). Each robot in the system has sensory capabilities allowing it to determine the location of other

* Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

robots in the environment, relative to its own location (each robot refers in fact to a *local coordinate system* that might be different from robot to robot). Each robot, when active, operates in `Look-Compute-Move` cycles: it determines the positions of the robots in the system (`Look`); this information is used to compute a destination point (`Compute`); the robot then moves towards the computed destination (`Move`); after the execution of a cycle, the robot may become temporarily inactive. Furthermore, the entities are *oblivious*: at the beginning of a cycle the robot has no recollection of computations and operations performed in previous cycles; that is, there is no persistent memory. This computational model is called OBLOT and it is a standard de-facto in the context of distributed computing by mobile entities [24]. In this model, the research effort has been on determining which problems can be solved by a swarm of such robots. Crucial for the solvability of a problem is the activation schedule of robots and the duration of their activities in each cycle. We consider the asynchronous setting (ASYNC), where robots do not have a common notion of time (which is possibly continuous), and the times when each robot is activated as well as the duration of each activity is decided by an adversary for each cycle.

Concerning the coordination of autonomous mobile robots, the main focus has been on the important class of *Pattern Formation* problems, where the robots are required to arrange themselves to form a given geometric shape (e.g., [17, 20, 22, 29, 30]). The *Arbitrary Pattern Formation* is a specific version that asks to determine from which initial configurations it is possible to form any specific but arbitrary geometric pattern given as input (e.g., [4, 9, 23]). In [11, 25], the so-called *Embedded Pattern Formation* problem was studied where the pattern to be formed is provided as a set of visible points in the plane. The general class includes also the *Gathering* problem requiring the robots to move to the same location, not decided in advance. This problem is of particular importance and has been extensively studied. It has been fully characterized in [15] (for a recent survey, see [21] and references therein). A slightly different model imposing robots to gather at some visible and predetermined points provided in the Euclidean plane has been also investigated and fully characterized, see [5–7].

In the continuous setting, the robots are assumed to be able to execute accurate movements in any direction and by any amount, even by infinitesimally small amounts. Hence, even in densely crowded situations, punctiform robots can maneuver avoiding collisions. Certain models also permit the robots to move along curved trajectories, in particular, the circumference of a circle. The correctness of the algorithms relies on the accurate execution of the movements. However, for robots with weak mechanical capabilities, it may not be possible to execute such intricate movements with precision. This motivates to consider robots moving in a grid-based terrain where the movements are restricted only along grid lines and only to a neighboring grid point in each step. Grid type floor layouts can be easily implemented in real-life robot navigation systems. From an algorithmic perspective, the restrictions imposed by model on the movements make it harder to solve problems that resulted to be easy in the continuous environment. In the grid environment, the most investigated types of formation problems are the Gathering problem [18] and Mutual Visibility problem [1], where a set of opaque robots have to form a pattern in which no three robots are collinear. The gathering problem has been investigated also in other specific graph topologies like trees [19], rings [16, 8], regular bipartite graphs [27], hypercubes [3], complete and complete bipartite [12, 13]. For a recent survey, see [10] and references therein. Few results concerning the general pattern formation problem on grids exist. Recently, the *Arbitrary Pattern Formation* for a set of oblivious asynchronous robots on the infinite grid in absence of any global coordinate system was first considered in [2]. Authors have shown that if the initial configuration is asymmetric, then the *Arbitrary Pattern Formation* problem is deterministically solvable by ASYNC robots starting from asymmetric configurations. They poses as an open problem that of investigating about what can be achieved from symmetric configurations. This leads to the problem addressed in this work: *how to break symmetries on grid based environments so that robots can later form any requested geometric pattern.*

Our contribution. We extend the concept of discretization of the Euclidean plane by considering regular tessellation graphs, that is square, triangular, and hexagonal grids. We assume very weak robots moving in this environment: they are asynchronous, oblivious, anonymous, silent, and fully disoriented. In this context, we consider the so-called *leader configurations*, that is the symmetric configurations in which it is possible to elect a leader and, as a consequence, it is possible to break the symmetry. However, breaking the symmetry by moving a leader robot is not a straightforward task due to the movement restrictions as all the adjacent nodes of the leader may be occupied. It may even happen that before obtaining the requested asymmetric configuration, most of the robots must be moved. Due to the asynchrony of robots, this fact greatly increases the difficulty of breaking the symmetry. We devise an algorithm called $\mathcal{A}_{break}$ able to solve the *Symmetry Breaking* problem on both the square and triangular grids. The algorithm is proposed so that it can be also combined with other modules (e.g., modules that are able to form some kind of pattern starting from any asymmetric configuration).

2 Basic notation and problem definition

We denote by $R = \{r_1, r_2, \ldots, r_n\}$ the set of robots forming the swarm under consideration.[1] The topology where robots are placed on is represented by a simple, undirected, and connected graph $G = (V, E)$, with vertex set V and edge set E. Given a function $\mu : R \to V$ that maps each robot to the vertex in G where the robot is placed, we call $C = (G, R, \mu)$ a *configuration*. A vertex $v \in V$ is said *occupied* if there exists $r \in R$ such that $\mu(r) = v$, *unoccupied* otherwise. A *multiplicity* occurs in any vertex $v \in V$ whenever there is more than one robot occupying v (i.e., when μ is not injective). With $mul(v)$ we denote the

[1] We recall that robots are anonymous and such a notation is used only for the sake of presentation, hence no algorithm can take advantage of names of elements in R.

multiplicity in v, that is the number of robots occupying v. As usual, $N(v)$ represents the set containing all the neighbors of the vertex v, that is all vertices adjacent to v; concerning robots, $N(r)$ contains all the robots "adjacent" to r, that is $N(r) = \{r' \in R : \mu(r') \in N(\mu(r))\}$. In our algorithm, in some cases, a robot is moved only when $N(r) = \emptyset$: accordingly, a robot r is said *blocked* if $N(r) \neq \emptyset$, *unblocked* otherwise.

Movements of robots and execution of an algorithm. The movement of the robots are restricted along the edges of the graph representing the environment in which robots operate, from one vertex to one of its neighboring vertices. Traditionally in discrete domains, robot movements are assumed to be *instantaneous*. This results in always perceiving robots on vertices and never on edges during `Look` phases. Hence, robots cannot be seen while moving, but only at the moment they may start moving or when they arrived.

In the Async scheduler, the activations of the robots determine specifically ordered time instants. Let $C(t)$ be the configuration observed by some robots at time t during their `Look` phase, and let $\{t_i : i = 0, 1, \ldots\}$, with $t_i < t_{i+1}$, be the set of all time instances at which at least one robot takes the snapshot $C(t_i)$. Since the information relevant for the computing phase of each robot is the order in which the different snapshots occur and not the exact time in which each snapshots is taken, without loss of generality we can assume $t_i = i$ for all $i = 0, 1, \ldots$. 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 some robot according to the result of the `Compute` phase as implemented by $\mathcal{A}$. Notice that this definition of execution works also for the other schedulers. Moreover, given an algorithm $\mathcal{A}$, in Async there exists more than one execution from $C(0)$ depending on the activation of the robots (which depends on the adversary).

Initially robots are inactive, but once the execution of any algorithm $\mathcal{A}$ starts there is no instruction to stop it, i.e., to prevent robots to enter their `LCM` cycles. Then, the *termination property* for $\mathcal{A}$ can be stated as follows: once robots have reached the required goal by means of $\mathcal{A}$, from there on robots can perform only the *nil* movement.

Configurations on tessellation graphs. In this work, we consider G as an infinite graph generated by a *plane tessellation*. A tessellation is a tiling of a plane with polygons without overlapping. A *regular* tessellation is a tessellation which is formed by just one kind of regular polygons of side length 1 and in which the corners of polygons are identically arranged. According to [26], there are only three regular tessellations, and they are generated by squares, equilateral triangles or regular hexagons (see Fig. 1). An infinite lattice of a regular tessellation is a lattice formed by taking the vertices of the regular polygons in the tessellation as the points of the lattice. A graph G is induced by the point set S if the vertices of G are the points in S and its edges connect vertices that are distance 1 apart. A *tessellation graph* of a regular tessellation is the infinite graph embedded into the Euclidean plane induced by the infinite lattice formed

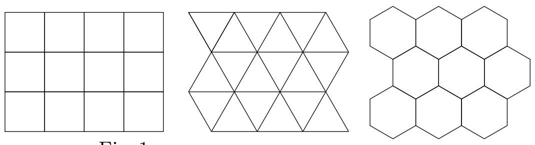
Fig. 1: Part of regular plane tessellations.

by that tessellation [28]. We denote by G_S (G_T and G_H, resp.) the tessellation graphs induced by the regular tessellations generated by squares (equilateral triangles and regular hexagons, resp.). In this work we consider configurations $C = (G, R, \mu)$ where $G \in \{G_S, G_T, G_H\}$.

Concerning any graph $G \in \{G_S, G_T, G_H\}$, it follows from the definition that G is regular, and hence by $deg(G)$ we denote the degree of each vertex. Notice that $deg(G)$ equals three, four, and six in G_H, G_S, and G_T, respectively. Any line parallel to an edge of G is called a *canonical line*, and the smallest angle formed by the available canonical lines is called the *canonical angle*. According to this notation, in G_S all the canonical lines have just two orientations and the canonical angle is of $90°$. In both G_T and G_H all the canonical lines have three orientations and the canonical angle is of $60°$. In the rest of the paper, given any tessellation graph G, by *hline* we mean any half-line starting from a vertex and coincident with any canonical line.

Configuration automorphisms and symmetries. 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, that is a permutation of the vertices of G that maps edges to edges and non-edges to non-edges. The set of all automorphisms of G, under the composition operation, forms a group called *automorphism group* of G and denoted by $\mathrm{Aut}(G)$. If $|\mathrm{Aut}(G)| = 1$, that is G admits only the identity automorphism, then G is said *asymmetric*, otherwise it is said *symmetric*. Two distinct vertices $u, v \in V$ are *equivalent* if there exists an automorphism $\varphi \in \mathrm{Aut}(G)$ such that $\varphi(u) = v$.

The concept of graph isomorphism can be extended to configurations in a natural way. Two configurations $C = (G, R, \mu)$ and $C' = (G', R', \mu')$ are isomorphic if there exists an isomorphism φ between G and G' that can be extended to obtain a bijection from R to R' such that two robots can be associated by φ only if they reside on equivalent vertices. Formally, if $\varphi(r) = r'$ then $\varphi(\mu(r)) = \mu'(r')$. In this way, analogously to the case of graph automorphism, an automorphism of a configuration $C = (G, R, \mu)$ is an isomorphism from C to itself, and the set of all automorphisms of C forms a group under the composition operation that we call automorphism group of C and denote as $\mathrm{Aut}(C)$. Moreover, if $|\mathrm{Aut}(C)| = 1$ we say that C is *asymmetric*, otherwise it is *symmetric*. Two distinct robots r and r' in a configuration (G, μ) are *equivalent* if there exists $\varphi \in \mathrm{Aut}(C)$ such that $\varphi(r) = r'$. Notice that, according to the definition, distinct robots in the same multiplicity are equivalent and hence each configuration with a multiplicity is symmetric. Also, note that $mul(u) = mul(v)$ whenever u and v are equivalent.

It can be observed that if r and r' are equivalent robots, no algorithm can distinguish between them. Hence, no algorithm can avoid the two equivalent ASYNC robots start the computational cycle simultaneously at a certain time t'. In such a case, there might be a so-called *pending move* (or *pending robot*), that is one of the two robots performs its entire computational cycle while the other has not started or not yet finished its Move phase. Formally, a robot r is pending in a configuration $C(t)$, if at time t robot r is active, has taken a snapshot $C(t') \neq C(t)$ with $t' < t$, and is planning to move or is moving with a non-nil trajectory. Clearly, any other robot r' that takes the snapshot $C(t)$ is not aware whether there is a pending robot r, that is it cannot deduce such a piece of information from the snapshot acquired in the Look phase. This fact greatly increases the difficulty to devise algorithms for ASYNC robots, and this holds in particular in symmetric configurations, where pending moves can be easily generated by the adversary. It follows that each time a formal and sound proof of the correctness of the algorithm must be provided, each algorithm must ensure to solve a general task by providing a *stationary* configuration: a configuration $C(t)$ is called stationary if there are no pending robots in $C(t)$.

Leader configurations and the Symmetry Breaking problem. Concerning the configurations addressed in this work, it is not difficult to see that any $C = (G, R, \mu)$, with $G \in \{G_S, G_T, G_H\}$, admits two types of automorphisms only: *reflections*, defined by a reflection axis which acts as a mirror; *rotations*, defined by a center and an angle of rotation. A configuration admitting only one reflection axis is called *reflective*, and a configuration admitting any rotation is called *rotational.* Notice that a configuration with two or more reflection axes is rotational.

It is well-known (e.g., see [31]) that no algorithm can break a symmetry among a group of two or more pairwise equivalent robots if it acts on that group only, even in the synchronous setting. In fact, since the algorithm cannot distinguish between them, any strategy defined by the algorithm will be applied by the adversary to all the considered robots. As a final result, the moved robots will remain symmetric in any possible obtained configuration. This implies that it is worth to address the problem of designing symmetry breaking algorithms only for special cases of symmetric configurations, as defined in the following.

Definition 1 (leader-configuration). *A configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T, G_H\}$, is a* leader configuration *if one of the following cases holds: (1) C is reflective, and there are one or more robots on the axis of reflection; (2) C is rotational, and there is one robot on the center of rotation.*

Notice that in any leader configuration there exists a robot which is equivalent to itself. In principle, this means that there could exist an algorithm that can move one of such robots to create an asymmetric configuration. We can now formalize the main problem addressed in this work.

Definition 2 (initial-configuration). *A configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T, G_H\}$, is an* initial configuration *if both the following conditions* *hold: (1) each robot is idle and placed on a different vertex, that is $mul(v) \leq 1$ for each $v \in V$; (2) C is a leader configuration.*

The set containing all the initial configurations is denoted by $\mathcal{I}$. The goal of the Symmetry Breaking (*SB*, for short) problem is to design any distributed algorithm $\mathcal{A}$ that, starting from any configuration $C \in \mathcal{I}$, guides the robots to form an asymmetric configuration C'. Formally, an algorithm $\mathcal{A}$ solves the *SB* problem for any configuration $C \in \mathcal{I}$ if, for each possible execution $\mathbb{E} : C = C(0), C(1), \ldots$ of $\mathcal{A}$, there exists a finite time instant $t^* > 0$ such that $C(t^*)$ is asymmetric and no robot moves after t^*, i.e., $C(t) = C(t^*)$ holds for all $t \geq t^*$.

3 Concepts and notation used by algorithm $\mathcal{A}_{break}$

Here we introduce some concepts and notation used in the proposed algorithm $\mathcal{A}_{break}$. They refer to any configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$.

Bounding parallelogram. We introduce the concept of *bounding parallelogram* $bp(R)$, defined as any parallelogram enclosing all robots, with sides parallel to two of the available grid line orientations, and with each pair of parallel sides as close together as possible. Since G_T or G_H admit canonical lines along three orientations, it can be observed that the bounding parallelogram of R is not unique on such topologies. In fact, there are up to three possible bounding rectangles (e.g., see Fig. 2). On G_S, $bp(R)$ is unique and corresponds to the well-known concept of *minimum bounding rectangle*. We denote by $h(bp(R))$ and $w(bp(R))$ the width and height of any $bp(R)$, respectively. Without loss of generality, we assume $h(bp(R)) \leq w(bp(R))$.

Binary strings associated to a configuration. Any algorithm addressing the *SB* problem needs to elect a leader among the robots in any initial configuration. If C is rotational, such a leader can be naturally identified with the robot occupying the center of rotation. Less obvious is how to identify a specific robot in reflective configurations. To this aim, in the following, we associate a binary string to any configuration so that from that string it is possible to elect a leader also in the case of initial reflective configurations.

Given any $bp(R)$, we associate a binary string to each *canonical corner* of $bp(R)$ (a canonical corner is a corner of the parallelogram that forms a canonical angle, e.g., corners A and C in Fig. 2). The string associated with a canonical corner A is defined as follows. Scan the finite tessellation enclosed by $bp(R)$ from A along $h(bp(R))$ (say, from A to B) and sequentially all canonical lines parallel to AB in the same direction. For each vertex v, put a 0 or 1 according to whether it is empty or occupied. Denote the obtained string as $s(AB)$. Being $h(bp(R)) = w(bp(R))$ in the example, from A it is also possible to obtain the string $s(AD)$, and hence four strings can be defined in total, two for the corner A and two for the corner C. Notice that if any two of these strings are equal, then the configuration is reflective or rotational.

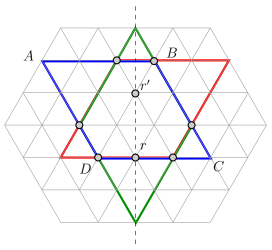

Fig. 2: Visualization of some notation used in the paper. An initial reflective configuration C defined on G_T: circles represent robots. It shows the three possible bounding parallelograms. They have all the same size. $LSS(R) =$ 0011000111011010 and it is generated in both the red and blue parallelograms. In the blue one, $LSS(R)$ is generated from the corner A along the side AD. Robot r is the pivot of the configuration.

Definition 3 **($LSS(R)$).** *Let $C = (G, R, \mu)$ be a configuration, with $G \in \{G_S, G_T\}$, and let S be the set containing all the binary strings associated to each canonical corner of $bp(R)$, for each $bp(R)$ with minimum height and, in case of ties, with minimum width. $LSS(R)$ denotes the* lexicographically smallest string *in S.*

It follows from the definition that $LSS(R)$ is unique, even when it is computed on symmetric configurations, where multiple $bp(R)$'s must be considered (cf. Fig. 2). By using $LSS(R)$, it is now possible to elect a leader, called *pivot*, in any initial reflective configuration $C = (G, R, \mu)$: the pivot is the median robot on the reflection axis of C (in case of ties, i.e. when the number of robots on the axis is even, the pivot is the median robot having the smallest position in $LSS(R)$). Concerning the example in Fig. 2, the represented configuration is reflective with two robots on the axis of reflection, the pivot is the robot denoted as r since its position in $LSS(R)$ is 8 whereas the position of the other is 10.

Strings generated from a robot. Given a robot r, the *strings generated from* r are the binary strings obtained in three steps, in order, as follows:

1. scan a hline that starts from the vertex $v = \mu(r)$ and stop when the last occupied vertex is reached: for each encountered vertex (v excluded) put 0 or 1 according to whether it is empty or occupied; if no occupied vertices are encountered, the empty string is returned;
2. repeat the previous step for each hline starting from the vertex $v = \mu(r)$, and insert all the obtained strings into a multiset $\mathcal{S}(r)$ - let Γ be the length of the longest string in $\mathcal{S}(r)$;
3. modify each string in $\mathcal{S}(r)$ by adding to the right of each string as many 0's as necessary to make the length of each string equal to $\Gamma + 1$.

Concerning configuration C_2 in Fig. 3, $\mathcal{S}(r)$ contains six strings, three equal to 110 and three equal to 010. Elements in $\mathcal{S}(r)$ are considered again as lexicographically ordered.

Definition 4. *Let S be a multiset containing some strings of $\mathcal{S}(r)$, and let* $\min(S)$ *and* $\max(S)$ *be the largest and smallest strings of S, respectively: we say that the strings in S are* almost-equal *if both the following conditions hold: (1) each string $s \in S$ is either equal to* $\min(S)$ *or* $\max(S)$*, and (2)* $\min(S)$ *can*

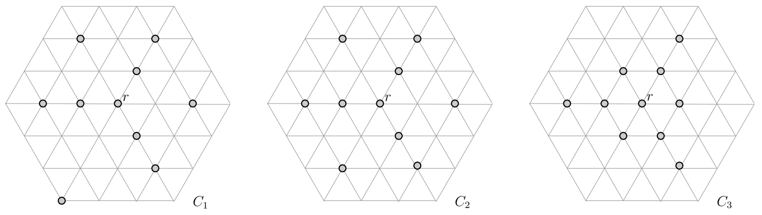

Fig. 3: Examples about notation *Compact*() and *Reduce*(): $C_2 = \mathit{Reduce}(C_1, r)$ and $C_3 = \mathit{Compact}(C_2, r) = \mathit{Compact}(C_1, r)$.

be made equal to $\max(S)$ *by just reversing one occurrence of the substring* 01 *in* $\min(S)$.

Given a robot r, if $\mathcal{S}(r)$ contains strings without any 1 we say that r has *free paths*. When r has no free paths, there could exist partitions of $\mathcal{S}(r)$ defined as follows:
• $\{S_1, S_2, \ldots, S_k\}$ is a *rotational-partition* of $\mathcal{S}(r)$ if: (1) there exists an integer $q > 1$ such that, for each set S_i, $|S_i| = q$ and the hlines corresponding to the string in S_i partition the plane into sectors of $360/q$ degrees each; (2) for each set S_i, the strings in S_i are almost-equal; (3) k is minimum.
• $\{S_1, S_2, \ldots, S_k\}$ is a *reflective-partition* of $\mathcal{S}(r)$ if: (1) there exists a line L such that, for each set S_i, $|S_i| = 2$ and L is the bisector of the hlines corresponding to the strings in S_i or L is coincident with both the hlines corresponding to the strings in S_i; (2) for each set S_i, the strings in S_i are almost-equal; (3) k is minimum.

As an example, consider robot r in configuration C_1 represented if Fig. 3: $\{S_1, S_2\}$ with $S_1 = \{1100, 1100, 1100\}$ and $S_2 = \{0100, 0100, 0010\}$ is a rotational-partition of $\mathcal{S}(r)$. Notice that, in the previous definition the value k ranges from one to three, and the latter occurs in the tessellation graph with the largest degree, i.e. G_T. If the strings in $\mathcal{S}(r)$ form a rotational-partition or a reflective-partition $\{S_1, S_2, \ldots, S_k\}$, then by notation $\mathit{Reduce}(C, r)$ we denote the configuration obtained from C by replacing, for each set S_i, each string $s \in S_i$ with the largest string $\max(S_i)$. By $\mathit{Compact}(C, r)$ we denote the configuration obtained from C by replacing each string $s \in \mathcal{S}(r)$ with its "compact version", that is the largest binary string containing the same number of 1's as s. Examples about notation *Compact*() and *Reduce*() are provided in the caption of Fig. 3.

4 Formalization of $\mathcal{A}_{break}$

In this section, we formalize the proposed algorithm $\mathcal{A}_{break}$ designed to solve the *SB* problem for any initial configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$, composed of n Async robots endowed with all the minimal capabilities recalled in the Introduction. We assume $n \geq 3$, since for $n = 1$ the *SB* problem is trivial and for $n = 2$ we get that C cannot be a leader configuration.

```
Algorithm: $\mathcal{A}_{break}$
Input: Leader configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$ and $R$ composed of $n \geq 3$
       ASYNC robots; external procedures IModule and FModule.

1  Call IModule ;
2  if $C \in$ aRot then
3      let $r$ be the robot that makes $C$ a-rotational (cf. Definition 5) ;
4      let $\mathcal{P} = \{S_1, S_2 \ldots, S_k\}$ be the rotational-regular partition of $\mathcal{S}(r)$ ;
5      call MakeSpace($r, \mathcal{P}$)
6  else if $C \in$ uRot then
7      the central robot $r$ of $C$ moves on one of its neighbors; if possible, $r$ selects a neighbor
         not belonging to an axis of reflection
8  else if $C \in$ fRot then
9      the central robot $r$ of $C$ moves on a neighbor belonging to any free path; if possible, $r$
         selects a neighbor not belonging to an axis of reflection
10 else if $C \in$ aDia then
11     let $r$ be the robot that makes $C$ a-diagonal (cf. Definition 7) ;
12     let $\mathcal{P} = \{S_1, S_2 \ldots, S_k\}$ be the diagonal-regular partition of $\mathcal{S}(r)$ ;
13     call MakeSpace($r, \mathcal{P}$)
14 else if $C \in$ uRef then
15     let $r$ be the robot on the axis of $C$, with $N(r) = \emptyset$, and having smallest position in
         LSS;
16     $r$ moves on one of its neighbors not belonging to the axis of reflection
17 else if $C \in$ fRef then
18     let $r$ be the robot on the axis of $C$ with free paths, and having smallest position in
         LSS ;
19     $r$ moves on a neighbor belonging to any free path
20 else if $C$ is asymmetric then
21     call FModule
```

Algorithm $\mathcal{A}_{break}$ makes use of three distinct procedures:[2] (1) Procedure *MakeSpace*, which is a procedure used in $\mathcal{A}_{break}$ "to make space around the central robot" by moving the robots that lie on the axes of symmetry so as to push them away from the center. (2) Procedure *IModule*, an external module taken as input. If $\mathcal{A}_{break}$ is simply used to solve the *SB* problem, then it corresponds to an empty procedure (e.g., no instructions contained). In case $\mathcal{A}_{break}$ is used as a breaking symmetry module for obtaining some more general algorithm $\mathcal{A}$, it can be used to check the termination property of $\mathcal{A}$. (3) Procedure *FModule*, an external module taken as input. If $\mathcal{A}_{break}$ is used to solve the *SB* problem, then it contains the following simple instruction: *each robot performs the nil movement*. Conversely, in case $\mathcal{A}_{break}$ is used as a breaking symmetry module for solving some general problem Π defined for leader or asymmetric configurations, then *FModule* corresponds to any algorithm for Π but for asymmetric configurations only.

Basically, algorithm $\mathcal{A}_{break}$ determines which class the input configuration belongs to, with respect to some classes that are formalized in what follows.

Definition 5 (a-rotational configuration). *A configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$, is called* almost-rotational *(a-rotational, for short) if there exists a robot $r \in R$ such that all the following conditions hold: (1) r is blocked*

[2] According to the LCM model, we assume that each robot terminates the execution of any algorithm or procedure as soon as it detects the move to be performed.

```
Procedure: MakeSpace
Input: Robot $r$ and a partition $\mathcal{P} = \{S_1, S_2 \ldots, S_k\}$ of $\mathcal{S}(r)$.

1 if there exist a multiset $S_i \in \mathcal{P}$ having different strings then
2     foreach $S_i \in \mathcal{P} : \min(S_i) \neq \max(S_i)$ do
3         foreach $s \in S_i : s = \max(S_i)$ do
4             let $r$ be the robot corresponding to the bit 1 in the substring "10" to be
                reversed in order to make $s$ equal to $\min(S_i)$ ;
5             $r$ moves so that $s$ becomes equal to $\min(S_i)$
6 else // for each $S_i$, the strings in $S_i$ are all the same
7     foreach $s \in S_i$ that starts with 1 do
8         let $r'$ be the robot corresponding to the 1 in the first occurrence of the substring
            "10" of $s$;
9         $r'$ moves away from $r$ along the hline corresponding to $s$
```

in C; (2) all strings in $\mathcal{S}(r)$ form a rotational-regular partition; (3) Reduce(C, r) is rotational and r is central in Compact(C, r).

Definition 6 (diagonal configuration). *An initial configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$, is called* diagonal *if it is reflective and its reflection axis does not coincide with any canonical line.*

Definition 7 (a-diagonal configuration). *A configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$, is called* almost-diagonal *(a-diagonal, for short) if there exists a robot $r \in R$ such that all the following conditions hold: (1) r is blocked in C; (2) all strings in $\mathcal{S}(r)$ form a reflective-regular partition; (3) Reduce(C, r) is diagonal and r is pivot in Compact(C, r).*

Robot r as in Definition 5 (Definition 7, resp.) is said the *robot that makes* C a-rotational (a-diagonal, resp.). Symbols aRot and aDia denote the classes containing all the a-rotational and a-diagonal configurations, respectively. Additional classes of configurations managed by $\mathcal{A}_{break}$ are the following:

– fRot denotes the class containing all the *free-rotational* (*f-rotational*, for short) configurations. A configuration C is free-rotational if it is rotational and its central robot r has free paths;
– uRot denotes the class containing all the *unblocked-rotational* (*u-rotational*, for short) configurations. A configuration C is u-rotational if it is rotational and its central robot r has no free paths, but $N(r) = \emptyset$;
– fRef denotes the class containing all the *free-reflective* (*f-reflective*, for short) configurations. A configuration C is free-reflective if it is reflective and there exists a robot r on the axis of C with free paths;
– uRef denotes the class containing all the *unblocked-reflective* (*u-reflective*, for short) configurations. A configuration C is u-reflective if it is reflective and each robot on its axis has no free paths, but there exists a robot r on the axis such that $N(r) = \emptyset$.

It can be observed that the above definitions give rise to sets that cover all the initial configurations in $\mathcal{I}$. In fact, (1) $\mathcal{I}$ can be partitioned into rotational and reflective configurations by definition; (2) rotational configurations are further partitioned into those with the central robot having free paths (i.e., f-rotational) and those with the central robot having no free paths - the latter are further divided into those with central robots unblocked (i.e., u-rotational) and those with central robot blocked (i.e., a-rotational); (3) similarly, reflective configurations are partitioned into the three classes of f-reflective, u-reflective, and a-reflective configurations. It is worth to note that $\mathcal{A}_{break}$ checks the membership of the input configuration C to the defined classes in a specific order, and this order is important for the correctness of the algorithm.

Theorem 1. *Algorithm $\mathcal{A}_{break}$ is able to solve the SB problem with respect to any initial configuration $C = (G, R, \mu)$ such that $G \in \{G_S, G_T\}$.*

Sketch of the Proof. If $C \in \mathsf{fRef} \cup \mathsf{uRef} \cup \mathsf{fRot} \cup \mathsf{uRot}$ it is possible to select just one robot to break the symmetry. In some cases, one move is enough, whereas in other cases several moves are necessary (e.g., $C \in \mathsf{uRot}$, the central robot moves to a neighbor, and the obtained configuration is in fRef). In these cases, the algorithm produces an asymmetric configuration without pending moves. Assume there exists a robot r that makes C a-rotational: both r and a rotational-regular partition of $\mathcal{S}(r)$ are passed as input to *MakeSpace*. Let $C(1)$ be any configuration generated according to the execution of *MakeSpace*. According to the hypothesis and to the move performed by the algorithm, it can be observed that $C(1)$ results to be in aRot too, and in particular the same robot r that was central in $C(0)$ is now the robot that makes $C(1)$ a-rotational. Hence, when $\mathcal{A}_{break}$ processes $C(1)$, the procedure moves exactly those robots that were not activated in $C(0)$ or pending in $C(1)$. This implies that all such robots are moved so that they will become stationary. Repeated calls to *MakeSpace* will finally push the robots forward until the robot r - the central one in $C(0)$ - becomes unblocked in an obtained configuration $C(t)$, for a finite $t > 0$. If $C \in \mathsf{aDia}$, the same analysis applies, but here the key property of the algorithm is that *MakeSpace* may produce an a-rotational configuration $C(1)$. We are able to show that in such a case the robot r that was pivot in C becomes the central robot in $C(1)$. Again, $\mathcal{A}_{break}$ correctly processes all the pending robots. □

5 Conclusion

We investigated the *Symmetry Breaking* problem in grid graphs. In this environment, breaking the symmetry by moving some leader robot is not a straightforward task due to the movement restrictions as all the adjacent nodes of the leader may be occupied. We have shown that it is possible solve the problem on both G_S and G_T graphs. The algorithm is proposed so that it can be also combined with other modules.

The most obvious open problem is to extend the proposed algorithm $\mathcal{A}_{break}$ to work also in the hexagonal grid G_H. $\mathcal{A}_{break}$ uses few geometric concepts, such as: bounding parallelogram, grid line, shortest path, and "moving along a line". Moving to hexagonal grids, G_H can be considered as a sub graph of G_T in which the center of the hexagons correspond to removed vertices. However by simply assuming the "presence" of the missing nodes and edges with respect to G_T, most of the geometric concepts introduced are still valid with the exception of "movement along a line". In fact, in G_H a robot cannot move along a line but it needs to move along the edges of successive hexagons.

As another possible future investigation, it would be worth to test whether it is possible to combine the proposed algorithm with that proposed in [2] to solve the *Arbitrary Pattern Formation* problem. This would bring us closer to characterizing such a problem on square grids. An advancement in this direction is presented in a very recent work [14].

References

1. Adhikary, R., Bose, K., Kundu, M.K., Sau, B.: Mutual visibility by asynchronous robots on infinite grid. In: Algorithms for Sensor Systems - 14th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS. Lecture Notes in Computer Science, vol. 11410, pp. 83–101. Springer (2018).
2. Bose, K., Adhikary, R., Kundu, M.K., Sau, B.: Arbitrary pattern formation on infinite grid by asynchronous oblivious robots. Theor. Comput. Sci. **815**, 213–227 (2020).
3. Bose, K., Kundu, M.K., Adhikary, R., Sau, B.: Optimal gathering by asynchronous oblivious robots in hypercubes. In: Proc. 20th Int.'l Symp. on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (Algosensors). LNCS, vol. 11410, pp. 102–117 (2019)
4. Bramas, Q., Tixeuil, S.: Arbitrary pattern formation with four robots. In: Proc. 20th Int.'l Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS). LNCS, vol. 11201, pp. 333–348. Springer (2018)
5. Cicerone, S., Di Stefano, G., Navarra, A.: Minimum-traveled-distance gathering of oblivious robots over given meeting-points. In: Proc. 10th Int.'l Symp. on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (Algosensors). LNCS, vol. 8847, pp. 57–72. Springer (2014)
6. Cicerone, S., Di Stefano, G., Navarra, A.: Minmax-distance gathering on given meeting-points. In: Proc. 9th Int.'l Conf. on Algorithms and Complexity (CIAC). LNCS, vol. 9079, pp. 127–139. Springer (2015)
7. Cicerone, S., Di Stefano, G., Navarra, A.: Gathering of robots on meeting-points: feasibility and optimal resolution algorithms. Distributed Computing **31**(1), 1–50 (2018)
8. Cicerone, S., Di Stefano, G., Navarra, A.: "Semi-Asynchronous": a new scheduler for robot based computing systems. In: Proc. 38th IEEE Int.'l Conf. on Distributed Computing Systems, (ICDCS). pp. 176–187. IEEE (2018)
9. Cicerone, S., Di Stefano, G., Navarra, A.: Asynchronous arbitrary pattern formation: the effects of a rigorous approach. Distributed Computing **32**(2), 91–132 (2019)
10. Cicerone, S., Di Stefano, G., Navarra, A.: Asynchronous robots on graphs: Gathering. In: Flocchini, P., Prencipe, G., Santoro, N. (eds.) Distributed Computing by Mobile Entities, Current Research in Moving and Computing, LNCS, vol. 11340, pp. 184–217. Springer (2019). https://doi.org/10.1007/978-3-030-11072-7_8
11. Cicerone, S., Di Stefano, G., Navarra, A.: Embedded pattern formation by asynchronous robots without chirality. Distributed Computing **32**(4), 291–315 (2019)

12. Cicerone, S., Di Stefano, G., Navarra, A.: Gathering synchronous robots in graphs: from general properties to dense and symmetric topologies. In: Proc. 26th Int.'l Colloquium on Structural Information and Communication Complexity (SIROCCO). LNCS, vol. 11639, pp. 170–184. Springer (2019)
13. Cicerone, S., Di Stefano, G., Navarra, A.: On gathering of semi-synchronous robots in graphs. In: Stabilization, Safety, and Security of Distributed Systems - 21st International Symposium, (SSS). LNCS, vol. 11914, pp. 84–98. Springer (2019). https://doi.org/10.1007/978-3-030-34992-9_7
14. Cicerone, S., Di Fonso, A., Di Stefano, G., Navarra, A.: Arbitrary Pattern Formation on Infinite Regular Tessellation Graphs - In: 22nd Int. Conf. on Distributed Computing and Networking, (ICDCN). ACM, (2021). To appear.
15. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Distributed computing by mobile robots: Gathering. SIAM J. on Computing **41**(4), 829–879 (2012)
16. D'Angelo, G., Navarra, A., Nisse, N.: A unified approach for gathering and exclusive searching on rings under weak assumptions. Distributed Computing **30**(1), 17–48 (2017)
17. Das, S., Flocchini, P., Santoro, N., Yamashita, M.: Forming sequences of geometric patterns with oblivious mobile robots. Distributed Computing **28**(2), 131–145 (2015)
18. Di Stefano, G., Navarra, A.: Gathering of oblivious robots on infinite grids with minimum traveled distance. Inf. Comput. **254**, 377–391 (2017)
19. Di Stefano, G., Navarra, A.: Optimal gathering of oblivious robots in anonymous graphs and its application on trees and rings. Distributed Computing **30**(2), 75–86 (2017)
20. Dieudonné, Y., Petit, F., Villain, V.: Leader election problem versus pattern formation problem. In: Proc. 24th Int.'l Symp. on Distributed Computing (DISC). LNCS, vol. 6343, pp. 267–281. Springer (2010)
21. Flocchini, P.: Gathering. In: Flocchini, P., Prencipe, G., Santoro, N. (eds.) Distributed Computing by Mobile Entities, Current Research in Moving and Computing, Lecture Notes in Computer Science, vol. 11340, pp. 63–82. Springer (2019). https://doi.org/10.1007/978-3-030-11072-7_4
22. Flocchini, P., Prencipe, G., Santoro, N., Viglietta, G.: Distributed computing by mobile robots: uniform circle formation. Distributed Comput. **30**(6), 413–457 (2017). https://doi.org/10.1007/s00446-016-0291-x
23. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. Theor. Comput. Sci. **407**(1-3), 412–447 (2008)
24. Flocchini, P., Prencipe, G., Santoro (Eds.), N.: Distributed Computing by Mobile Entities, Current Research in Moving and Computing, LNCS, vol. 11340. Springer (2019). https://doi.org/10.1007/978-3-030-11072-7
25. Fujinaga, N., Yamauchi, Y., Ono, H., Kijima, S., Yamashita, M.: Pattern formation by oblivious asynchronous mobile robots. SIAM J. Computing **44**(3), 740–785 (2015)
26. Grünbaum, B., Shepard, G.C.: Tiling and Patterns. W. H. Freeman & Co., New York (1987)
27. Guilbault, S., Pelc, A.: Gathering asynchronous oblivious agents with local vision in regular bipartite graphs. Theor. Comput. Sci. **509**, 86–96 (2013)
28. Ionascu, E.J.: Half domination arrangements in regular and semi-regular tessellation type graphs. Math **abs/1201.4624v1** (2012), https://arxiv.org/abs/1201.4624v1

29. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM J. Comput. **28**(4), 1347–1363 (1999)
30. Yamashita, M., Suzuki, I.: Characterizing geometric patterns formable by oblivious anonymous mobile robots. Theor. Comput. Sci. **411**(26-28), 2433–2453 (2010)
31. Yamauchi, Y.: Symmetry of anonymous robots. In: Flocchini, P., Prencipe, G., Santoro, N. (eds.) Distributed Computing by Mobile Entities, Current Research in Moving and Computing, Lecture Notes in Computer Science, vol. 11340, pp. 109–133. Springer (2019). https://doi.org/10.1007/978-3-030-11072-7_6