IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 29, 2021, accepted October 25, 2021, date of publication October 27, 2021,
date of current version November 8, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3123510

Breaking Symmetries on Tessellation Graphs via Asynchronous Robots: The Line Formation Problem as a Case Study

SERAFINO CICERONE
Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Via Vetoio, 67100 L'Aquila, Italy
e-mail: serafino.cicerone@univaq.it

ABSTRACT 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. This class of problems has been extensively studied in the continuous environment (e.g., the Euclidean plane), whereas few results exist when robots move in a discretization of the plane, like infinite grids. In this environment, to form any pattern, the problem of breaking symmetries 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. 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 the problem. We assume very weak robots moving on any regular tessellation graph as a discretization of the Euclidean plane, and we devise an algorithm $\mathcal{A}_{break}$ able to solve the *Symmetry Breaking* problem on both the square and triangular grids. It is important to note that $\mathcal{A}_{break}$ is proposed so that it can be used as a module for solving more general problems. As a case study, we use $\mathcal{A}_{break}$ to deal with the *Line Formation* problem, where $n \geq 3$ robots must arrange themselves to occupy n contiguous vertices along a grid line. In this respect, we first provide an algorithm $\mathcal{A}_{LF^-}$ able to partially solve this problem (it works with configurations in which it is not necessary to break symmetries), and then we show how $\mathcal{A}_{break}$ and $\mathcal{A}_{LF^-}$ can be combined to form $\mathcal{A}_{LF}$. We provide a complete characterization of the solvability of the *Line Formation* problem on the considered topologies by showing that $\mathcal{A}_{LF}$ solves the problem in each configuration where this is possible.

INDEX TERMS Distributed algorithms, mobile robots, asynchrony, pattern formation, symmetry, grid graphs.

I. INTRODUCTION

The coordination of autonomous mobile entities has long been the 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.

These entities, often 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 robots in the environment, relative to its own location (in fact, each robot refers 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`), it uses this information to compute a trajectory toward destination point (`Compute`), and then it moves along the computed trajectory towards the destination point (`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 associate editor coordinating the review of this manuscript and approving it for publication was Azwirman Gusrialdi.

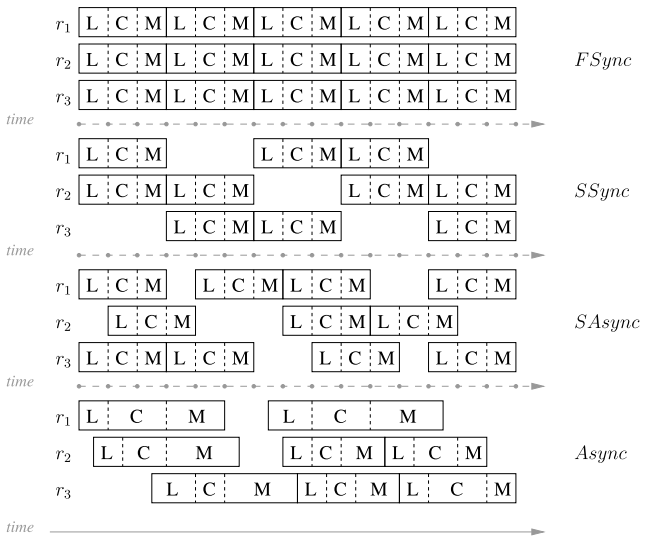

FIGURE 1. The execution model of computational cycles for each of FSYNC, SSYNC, SASYNC, and ASYNC robots. The inactivity of robots is implicitly represented by empty time periods.

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 one of the most investigated within distributed computing [26]. With respect to this model, the research effort is mainly to determine what problems can be solved by a swarm of such limited robots.

Crucial for the solvability of a problem is the activation schedule of robots and the duration of their activities in each cycle. In the literature, different characterizations of the environment have been considered according whether robots are fully-synchronous, semi-synchronous (cf. [34], [35], [37]), semi-asynchronous (cf. [15]) or asynchronous (cf. [3], [9], [10], [12], [19], [25], [28]). These synchronization models are illustrated in Figure 1 and defined as follows:

- *Fully-Synchronous* (FSYNC): All robots are always active, continuously executing their LCM cycles in a synchronized way. Hence the time can be logically divided into global rounds. In each round, each robot obtains a snapshot of the environment computed on the basis of the obtained snapshot, and then perform the computed move.
- *Semi-Synchronous* (SSYNC): It coincides with the FSYNC model, with the only difference that not all robots are necessarily activated in each round.
- *Semi-Asynchronous* (SASYNC): Robots are activated independently. Like in FSYNC or SSYNC, the duration of each phase is assumed to be always the same. Differently from FSYNC or SSYNC, two activated robots can be in different phases even though phases are synchronized.
- *Asynchronous* (ASYNC): Robots are activated independently, and the duration of each phase is finite but unpredictable. As a result, robots do not have a common notion of time.

In ASYNC, the amount of time to complete a full LCM-cycle is assumed to be finite but unpredictable. Moreover, in the SSYNC, SASYNC, and ASYNC cases it is usually assumed the existence of an *adversary* determining the timing of the computational cycles. Such timing is assumed to be *fair*, that is, each robot performs its LCM-cycle within finite time and infinitely often.

When robots move in a continuous environment like the Euclidean Plane, they are often viewed as points (they are dimensionless), and more than one robot can occupy the same location at the same time; when this occurs, we say that there is a *multiplicity*. It is often assumed (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. [17], [32]). During the Look phase, robots can perceive multiplicities, and this capability can 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.

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., [15], [19], [22], [24], [34], [35]). 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., [6], [10], [25]). In [12], [27], 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 of Pattern Formation includes also the *Gathering* problem, where robots are all required to move to the same location, not decided in advance. This problem is of particular importance and has been extensively studied when robots move in the Euclidean plane. In this environment, it has been fully characterized in [17] (for a recent survey, see [23] 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 [9].

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 manoeuvre 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 rely 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 on 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 (e.g., see industrial Automated Guided Vehicles [2] and Coverage Path Planning [33]). From an algorithmic perspective, the restrictions imposed by the model on the movements make it harder to solve problems that resulted to be easy in the continuous environment. In the square grid environment, the most investigated types of formation problems are the Gathering problem [20] and the 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 [21], rings [15], [18], regular bipartite graphs [30], hypercubes [5], complete and complete bipartite [13], [14]. For a recent survey, see [11] and references therein.

Few results concerning the general pattern formation problem in grids exist. Recently, the *Arbitrary Pattern Formation* for a set of oblivious asynchronous robots on the infinite square grid in the absence of any global coordinate system was first considered in [4]. The authors have shown that if the initial configuration is asymmetric, then the *Arbitrary Pattern Formation* problem is deterministically solvable by ASYNC robots. This result has been recently extended in [8] to triangular and hexagonal grids, also allowing multiplicities in the pattern to be formed. To further extend these results to the more general *Pattern Formation*, authors in both [4] and [8] posed the open problem of investigating what can be achieved from symmetric configurations. This led to the *Symmetry Breaking* problem addressed in this work: *how to break symmetries in grid-based environments so that robots can eventually form any requested geometric pattern.*

A. CONTRIBUTION

As in [8], to model the discrete environment in which robots move, we consider any regular tessellation graph, 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 (but they are equipped with the global strong multiplicity detection capability). In this context, we consider the so-called *leader configurations*, that is the symmetric configurations [1] in which it is possible to elect a leader and, as a consequence, break the symmetry by moving the leader robot. However, breaking the symmetry by moving the 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 required asymmetric configuration, most of the robots must be moved. Along with the asynchrony of robots, this fact greatly increases the difficulty of designing an algorithm able to break symmetries.

We devise an algorithm called $\mathcal{A}_{break}$ able to solve the Symmetry Breaking problem on both the square and triangular grids. As a further contribution, the proposed algorithm is designed so that it can be also combined with other modules. As a case study, we apply $\mathcal{A}_{break}$ to solve the *Line Formation* problem, where $n \geq 3$ robots must arrange themselves in order to occupy n contiguous vertices along a grid line. In this respect, we first provide an algorithm $\mathcal{A}_{LF^-}$ able to partially solve this problem (it works with configurations where it is not necessary to break symmetries), and then we show how $\mathcal{A}_{break}$ and $\mathcal{A}_{LF^-}$ can be combined to form $\mathcal{A}_{LF}$. We provide a complete characterization of the solvability of the *Line Formation* problem on the considered topologies by showing that $\mathcal{A}_{LF}$ solves the problem in each configuration where this is possible.

B. ORGANIZATION

The rest of the paper is organized as follows. In Section II, some basic definitions and a formal description of the symmetry breaking problem are presented. Section III highlights some of the key difficulties that make the symmetry breaking problem challenging. It also provides insight into our algorithmic design choices and a high-level intuition of the proposed algorithm. Section IV contains a specific notation used by the algorithm whereas Section V provides its formalization and correctness. The case study concerning the Line formation problem is contained in Section VI. There, notation, formalization, and correctness are all provided. Finally, Section VII reports some concluding remarks.

II. 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. [2] The topology where robots are placed 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 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 the algorithm proposed in this work, a robot moves in order to break a symmetry only when all its neighborhood is empty. This motivates the following terminology: a robot r is *blocked* if $N(r) \neq \emptyset$, *unblocked* otherwise.

[1] In the literature, the definition of leader configurations also includes asymmetric configurations. Here we consider a more rigorous version because we are interested in the symmetry breaking problem, which does not apply in the case of asymmetric configurations.

[2] 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.

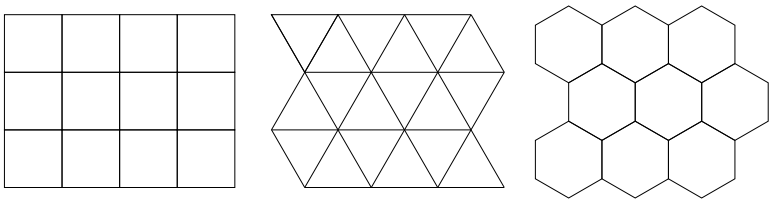

FIGURE 2. Part of regular plane tessellations.

A. MOVEMENTS OF ROBOTS AND EXECUTION OF AN ALGORITHM

The movement of the robots is 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 the edges during `Look` phases. Hence, robots cannot be seen by other robots while moving, but only at the moment they may start moving or when they arrived. Notice that during an `LCM` cycle a robot traverses only one edge.

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 snapshot 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 according to movements of robots as dictated by the `Compute` phase implemented by $\mathcal{A}$. This definition of execution works for all the four schedulers, but with the following remark: in ASYNC, $C(t + 1)$ can be generated by a movement planned in $C(t')$, with $t' \ll t$; in FSYNC or SSYNC, $C(t + 1)$ depends on movements planned in $C(t)$ only; in the SASYNC case, $C(t + 1)$ depends on movements planned in $C(t)$ or $C(t - 1)$. Moreover, given an algorithm $\mathcal{A}$, in ASYNC (but also in SSYNC and SASYNC) there exists more than one execution from $C(0)$ according to 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* of $\mathcal{A}$ can be stated as follows: once robots have reached the required goal by means of $\mathcal{A}$, from there on robots perform only the *nil* movement.

B. 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 that 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 [29], there are only three regular tessellations, and they are generated by squares, equilateral triangles or regular hexagons (see Fig. 2). 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 by that tessellation [31]. 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 $C = (G, R, \mu)$ with $G \in \{G_S, G_T, G_H\}$, but notice that most of the provided results hold for configurations in G_S and G_T only.

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 90°. In both G_T and G_H all the canonical lines have three orientations and the canonical angle is 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.

C. 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 is 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 of robots 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' are *equivalent* in a configuration C 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 according to a configuration automorphism.

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 $C(t') \neq C(t)$, $t' < t$, and is planning to move or is performing a non-nil movement. 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. Said that it is worth remarking that 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 $C(t)$. The request for generating a stationary configuration also holds before the termination. In fact, when the algorithm solves the general task by subdividing it into sub-tasks, it must ensure that all robots are aware that a sub-task has been performed (i.e., a particular kind of stationary configuration is generated) in order to proceed with another one in a controlled way. This is mandatory whenever formal and solid proof of the algorithm's correctness has to be provided.

D. LEADER CONFIGURATIONS AND THE SYMMETRY BREAKING PROBLEM

Concerning the configurations addressed in this work, it is not difficult to see that each $C = (G, R, \mu)$, 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. In a rotational configuration C, $\rho(C)$ denotes the "angle of rotation", which is the smallest angle for which the configuration can be rotated to coincide with itself. For configurations defined on G_S, $\rho(C) \in \{90°, 180°\}$, whereas for both G_T and G_H, $\rho(C) \in \{60°, 120°, 180°\}$. We say that a rotational configuration C is of:

- type 1, when the center of rotation is on a vertex of G;
- type 2, when the center of rotation is on a median point of an edge of G;
- type 3, when the center of rotation is on the center of any regular polygon of the tessellation forming G.

It is well-known (e.g., see [36]) 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 addressing 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 is one or more robots on the axis of reflection; (2) C is rotational of type 1, and there is one robot on the center of rotation.*

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 if, for each configuration $C \in \mathcal{I}$ and 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 $t \geq t^*$.

III. TECHNICAL DIFFICULTIES AND HIGH-LEVEL IDEAS

We now highlight some of the key difficulties that make the *SB* problem interesting. In doing so, we provide insights into

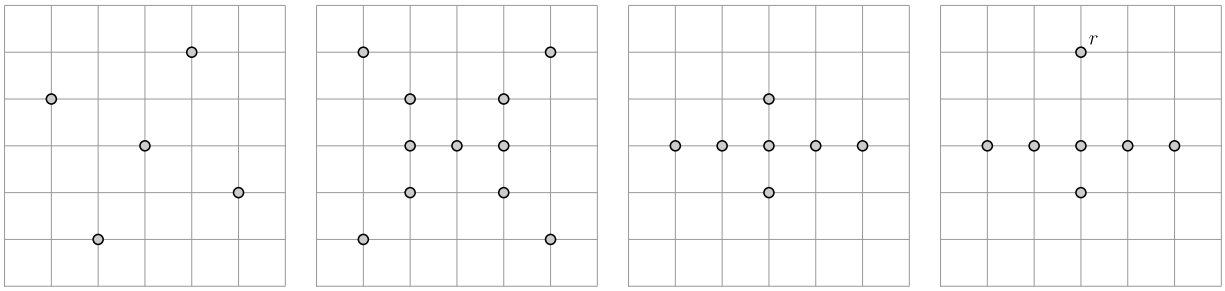

FIGURE 3. Representation of some initial configurations defined on G_S: from left to right, in the text they are referred as C_1, C_2, C_3, and C_4.

our algorithmic design choices and a high-level intuition of our algorithm, though some key details are elaborated upon only in the respective sections.

As highlighted in the Introduction, a fundamental difficulty behind any algorithm for the *SB* problem is due to the environment in which robots move: graphs. In fact, when robots move in the Euclidean plane, the typical approach is the following. Given a configuration of robots placed in the Euclidean plane, assume that P is the set containing all the robots' positions, and let $SEC(P)$ and $c(P)$ denote the smallest enclosing circle of P and its center, respectively. If the configuration is rotational and initial, then there is a single robot r on $c(P)$. To solve the *SB* problem, let δ be the distance between r and any other robot $r' \neq r$ closest to r, that is $\delta(r) = \min_{r' \in P}\{d(r, r') : r' \neq r\}$. To solve *SB*, it is enough to move r along a segment $[r, t]$ toward any point t such that $d(c(P), t) = \delta(r)/2$. Notice that: (1) since P is finite and the robots are placed on the Euclidean plane, there is certainly a segment as required, and (2) even if the configuration P changes into an asymmetric one as soon as r leaves $c(P)$, it is important that the robot reaches the defined target so that the algorithm can correctly guarantee the termination property.[3] A similar (and simple) approach to solve *SB*, also holds in the case of a reflective configuration with robots on the axis.

If we consider configurations defined on graphs, solving the *SB* problem may become a really complex task. Consider the configurations represented in Fig. 3:

- In $C_1 = (G_T, R, \mu)$, the central robot r is not blocked, that is the vertices in $N(\mu(r))$ are *all unoccupied*. In this case, to solve *SB*, it is enough to move r to any vertex in $N(\mu(r))$: as C_1 is rotational without axes of reflection, the obtained configuration is stationary and asymmetric.
- C_2 is still an initial rotational configuration, but it has two axes of reflection. The central robot r has a *free path*, i.e., it can freely move along one axis of reflection. In this case, solving *SB* simply requires moving r along the free path, thus generating an initial reflective configuration. In the obtained configuration, the same robot r can still move along the same direction until it reaches a position such that it becomes *unblocked*, that is $N(r) = \emptyset$ holds. Finally, moving r on one of such neighbors leads solving the problem. Notice that this approach never creates pending robots, so the obtained configurations are always stationary and the termination property is guaranteed.
- C_3 shows a more complex situation. There, the central robot r is blocked. To solve the *SB* problem, an algorithm could move the robots in $N(r)$ out of the axes of symmetry, and this would certainly lead having unoccupied neighbors for r. But, the adversary can create pending robots, and this would affect the termination property. Additionally, this approach is not always feasible, since even all robots in $N(r)$ could be blocked as well.

In contrast to the strategy discussed with respect to C_3 in Fig. 3, a more robust approach to *make space* around the central robot is moving the robots that lie on the axes of symmetry so as to push them away from the center. For example, in C_3 all four robots on the axes, those furthest from the center, should move along the axis. Once all the selected robots on the axis have moved, subsequent robots on the same axis should also make the same move, and thus recursively until the central robot becomes unblocked.

Any algorithm applying the latter approach has to cope with the following main issue: the adversary may not activate some robots or may delay the LCM cycle of others so that pending robots are created. This leads to non-stationary configurations where, due to the obliviousness property, the algorithm must detect possible pending robots and finalize their moves before proceeding further.

For example, when the algorithm processes the configuration C_4 in Fig. 3, before considering it a stationary reflective configuration, it should check if the configuration may have been generated by previous executions (e.g., it may have been generated from C_3 and have up to three pending robots). In that case, the algorithm must: (1) consider the configuration as *almost rotational* instead of rotational, and (2) accordingly, identify all possible robots that should have moved in the previous configuration and consequently complete their movement. Of course, the fewer robots they move in each move, the easier it is to have control over the termination of each phase.

[3]This may require that r uses more than one LCM cycle, since in the Euclidean plane the robots' movements are not rigid, and the adversary may stop the robots as soon as it performed a minimum guaranteed distance $\nu > 0$.

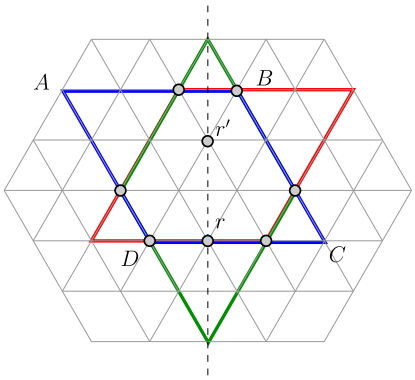

FIGURE 4. Visualization of some notation used in the paper. An initial reflective configuration C defined on G_T: circles represent robots. It shows the three possible bounding parallelograms. They have all the same size. $LSS(R) = 0011000111011010$ and it is generated in both the red and blue parallelograms. In the blue one, $LSS(R)$ is generated from corner A along the side AD. Robot r is the pivot of C.

Notice that all the previous considerations made about rotational configurations can be replicated in a similar way to reflective configurations. Finally, we remark that each algorithm designed to solve *SB* cannot create any multiplicity: the presence of a multiplicity in a configuration makes that configuration symmetric. Unfortunately, this kind of symmetry cannot be broken and hence *SB* cannot be solved.

IV. 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 $G \in \{G_S, G_T\}$.

A. BOUNDING PARALLELOGRAM

As in [8], we consider 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 parallelograms (e.g., see Fig. 4). 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 $h(bp(R)) \leq w(bp(R))$.

B. BINARY STRINGS ASSOCIATED WITH 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 (for this purpose, we use the example given in Fig. 4).

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. 4). 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 corner A and two for corner C. Notice that if any two of these strings are equal, then the configuration is reflective or rotational.

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 with 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. 4). By using $LSS(R)$, it is now possible to elect a leader, called *pivot*, in any initial reflective configuration.

Definition 4: *Let $C = (G, R, \mu)$ be any initial reflective configuration. The* pivot *of C 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. 4, 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 r' is 10.

C. STRINGS GENERATED FROM A ROBOT

Given a robot r, the *strings generated from r* are the binary strings obtained in three steps, performed 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 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_1' in Fig. 5, $\mathcal{S}(r)$ contains six strings, three equal to 110 and three equal to 010. Also in $\mathcal{S}(r)$ we consider the elements as lexicographically ordered.

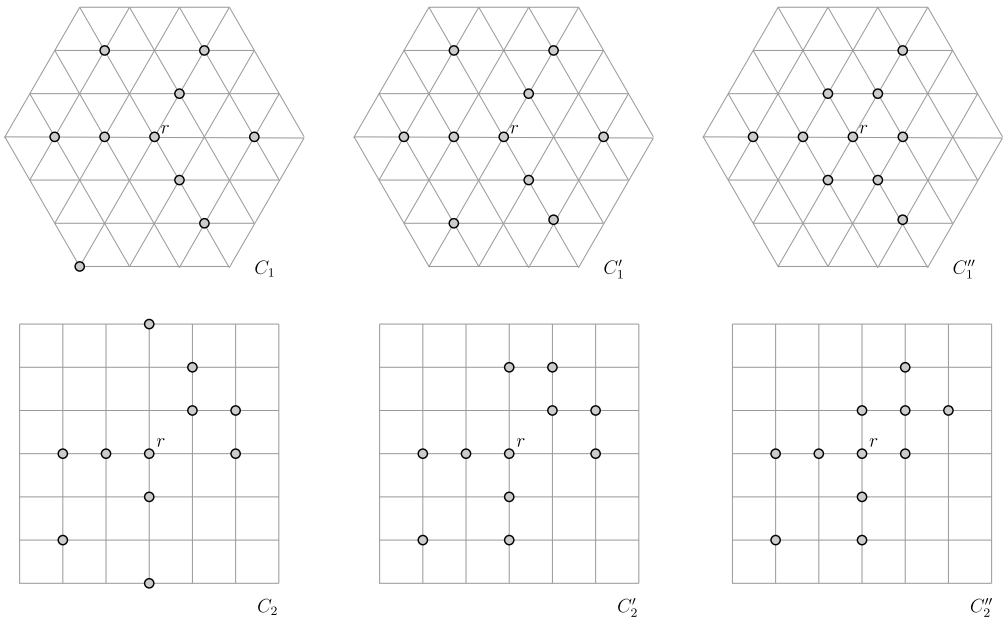

FIGURE 5. C_1 is almost-rotational (cf. Definition 6) and C_2 is almost-diagonal (cf. Definition 8). Notice that $C_1' = Reduce(C_1, r)$ and $C_1'' = Compact(C_1', r) = Compact(C_1, r)$; moreover, $C_2' = Reduce(C_2, r)$ and $C_2'' = Compact(C_2', r) = Compact(C_2, r)$.

Definition 5: 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* $\max(S)$*;*
2) $\min(S)$ *can 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* (e.g., the central robot in both configurations C_1 and C_2 shown in Fig. 3 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. 5: $\{S_1, S_2\}$ 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, namely 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 $Reduce(C, r)$ denotes the configuration obtained from C by replacing each string $s \in S_i$ with the largest string $\max(S_i)$, for each set S_i. By $Compact(C, r)$ we denote the configuration obtained from C by replacing each string $s \in \mathcal{S}(r)$ with its "compact version", which is the largest binary string containing the same number of 1's as s. For instance, by considering the configurations represented in Fig. 5, notice that $C_1' = Reduce(C_1, r)$ and $C_1'' = Compact(C_1', r) = Compact(C_1, r)$.

V. FORMALIZATION AND CORRECTNESS OF $\mathcal{A}_{break}$

In this section we formalize $\mathcal{A}_{break}$, an algorithm designed to solve the *SB* problem for any configuration $C = (G, R, \mu) \in \mathcal{I}$, 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.

The pseudo-code of $\mathcal{A}_{break}$ is formalized in Algorithm. 1. It makes use of three distinct procedures:

- Procedure *MakeSpace*, whose pseudo-code is given in Algorithm 2. It is an internal procedure used "to make space around the central/pivot robot r". The space is created by pushing away the robots that lie on the hlines that start from the vertex $v = \mu(r)$.
- Procedures $\mathcal{I}_{Mod}$ and $\mathcal{F}_{Mod}$, two external modules taken as input by $\mathcal{A}_{break}$. If $\mathcal{A}_{break}$ is used to solve the *SB* problem only, then both $\mathcal{I}_{Mod}$ and $\mathcal{F}_{Mod}$ contain the following simple instruction: *each robot performs the nil movement*. Conversely, $\mathcal{A}_{break}$ can be used as a breaking symmetry module when some general problem Π is

defined for both leader or asymmetric configurations. In such a case, $\mathcal{I}_{Mod}$ is responsible for checking the termination property of $\mathcal{A}$, and $\mathcal{F}_{Mod}$ corresponds to any algorithm $\mathcal{A}$ that solves Π but for asymmetric configurations only. A specific example of this approach will be provided in Section VI.

Moreover, with respect to the provided pseudo-code the following assumptions hold:

- If no destination is computed (like for instance when $\mathcal{F}_{Mod}$ is executed), the current `LCM` cycle is terminated with an empty move;
- when any destination is computed, the execution of the algorithm is terminated within the current `LCM` cycle and then the move is performed.

To describe the behavior of $\mathcal{A}_{break}$, consider that all the initial configurations in $\mathcal{I}$ can be divided into some classes (formalized in what follows according to the highlights and key difficulties described in Section III). Basically, algorithm $\mathcal{A}_{break}$ first determines which class the input configuration belongs to and consequently performs a dedicated move. Let us start by formally define the considered classes of configurations.

Definition 6 (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 in C;*
2) *there exists a rotational-partition for $\mathcal{S}(r)$;*
3) *Reduce(C, r) is rotational of type 1 and r is central in Compact(C, r).*

In Fig. 5, C_1 provides an example of a-rotational configuration.

Definition 7 (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.*

The following definition introduces the class of *almost-diagonal* configurations. It uses the notion of pivot as given in Definition 4.

Definition 8 (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) *there exists a reflective-partition for $\mathcal{S}(r)$;*
3) *Reduce(C, r) is diagonal and r is pivot in Compact(C, r).*

In Fig. 5, C_2 provides an example of a-diagonal configuration.

Robot r as in Definition 6 (Definition 8, resp.) is said the *robot that makes C* a-rotational (a-diagonal, resp.).

Lemma 9: There exists a unique robot that makes a configuration C almost-rotational (almost-diagonal, respectively).

Proof: We first prove the statement concerning almost-rotational configurations.

If C is rotational, r is unique since it is the central robot in the configuration. By contradiction, assume that C is not rotational and there are two distinct robots r' and r'' that both satisfy all the three conditions in Definition 6. We analyze two cases according whether r' and r'' are on a same canonical line or not.

Assume that r' and r'' are both on a same canonical line L. Since there exists a rotational-partition for r', there must be robots in L. Moreover, according to the definition of $\mathcal{S}(r')$, the two strings generated form r' and computed along the hlines coincident with L contains the same number of 1's. This means that on L there is an odd number of robot and r' is the median robot in L. Of course, it is not possible that also r'' is median in L and hence we get a contradiction.

Assume that r' and r'' are not on a same canonical line. This implies that r' is not "compacted" to get $Compact(C, r'')$ (and the same for r'' in $Compact(C, r')$). Hence, the relative position of r' and r'' in C is maintained in both $Compact(C, r')$ and $Compact(C, r'')$. Since r' is central in $Compact(C, r')$, then there must exist a robot r_1'' equivalent to r'' in $Compact(C, r')$. But now r_1'' must have an equivalent robot r_2'' in $Compact(C, r'')$, with the distance between r'' and r_1'' larger than the distance between r' and r_1''. Again, the last identified robot r_2'' need an equivalent robot r_3'' at a larger distance. We get a contradiction since this identification process would require infinitely many robots in C.

Similar arguments as above can be used to prove that there exists a unique robot r that makes C an almost-diagonal configuration. □

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:

- 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 there exists a robot r on the axis such that $N(r) = \emptyset$. As an example, see C_4 in Fig. 3.
- fRef denotes the class containing all the *free-reflective* (*f-reflective*, for short) configurations. A configuration C is free-reflective if it is reflective but not u-reflective. The name is motivated by the fact that in such configurations the robots on the axis and closest to $bp(R)$ have free paths. As an example, consider again C_4 in Fig. 3 but now assume the existence of a robot r', just below r, that makes the configuration not u-reflective.
- uRot denotes the class containing all the *unblocked-rotational* (*u-rotational*, for short) configurations. A configuration C is u-rotational if it is rotational of type 1 and its central robot r has free paths or $N(r) = \emptyset$. As an example, see C_1 in Fig. 3.

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 noting that $\mathcal{A}_{break}$ checks the membership of the input configuration C to the defined classes in a specific order. We will see that this order is important to assess the correctness of the algorithm.

Algorithm 1 $\mathcal{A}_{break}$

Input: A leader configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$ and R composed of $n \geq 3$ ASYNC robots; external procedures $\mathcal{I}_{Mod}$ and $\mathcal{F}_{Mod}$.
Output: A configuration solving *SB* with respect to the input configuration C

1: Call $\mathcal{I}_{Mod}$
2: **if** $C \in$ aRot **then**
3: let r be the unique robot that makes C a-rotational (cf. Definition 6 and Lemma 9)
4: let $\mathcal{P} = \{S_1, S_2 \ldots, S_k\}$ be the rotational-partition of $\mathcal{S}(r)$
5: call *MakeSpace*$(r, \mathcal{P})$
6: **if** $C \in$ uRot **then**
7: the central robot r moves on one of its unoccupied neighbors
8: **if** $C \in$ aDia **then**
9: let r be the unique robot that makes C a-diagonal (cf. Definition 8 and Lemma 9)
10: let $\mathcal{P} = \{S_1, S_2 \ldots, S_k\}$ be the diagonal-partition of $\mathcal{S}(r)$
11: call *MakeSpace*$(r, \mathcal{P})$
12: **if** $C \in$ fRef **then**
13: let r be the robot on the axis of C closest to $bp(R)$; in case of ties, consider the one having smallest position in *LSS*
14: r moves along any free path so that to increase its distance from the center of $bp(R)$
15: **if** $C \in$ uRef **then**
16: let r be the robot on the axis of C, with $N(r) = \emptyset$, and having smallest position in *LSS*
17: r moves on one of its neighbors not belonging to the axis of reflection
18: Call $\mathcal{F}_{Mod}$

Algorithm 2 *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: **for all** $S_i \in \mathcal{P} : \min(S_i) \neq \max(S_i)$ **do**
3: **for all** $s \in S_i : s = \max(S_i)$ **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**
7: **for all** $s \in \mathcal{S}(r)$ 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

Correctness: The following lemma deals with the correctness of $\mathcal{A}_{break}$ with respect to "simple" input configurations only, that is configurations not belonging to aDia $\cup$ aRot. When one of them is processed by $\mathcal{A}_{break}$, the algorithm is able to solve the *SB* problem by always producing stationary configurations throughout the execution.

Lemma 10: Let C be an initial configuration that does not belong to either aDia *or* aRot. *Then,* $\mathcal{A}_{break}$ *is able to solve the SB problem with respect to the input configuration C.*

Proof: We prove that for each possible execution $\mathbb{E} : C = C(0), C(1), \ldots$ of $\mathcal{A}_{break}$, 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 $t \geq t^*$. The proof proceeds by analyzing the different classes where $C(0)$ may belong to.

Let us assume that in $C(0)$ the test at Line 15 is evaluated true. Hence, $C \in$ uRef. By definition, C is reflective and there exists a robot r on the axis such that $N(r) = \emptyset$. In this situation, the algorithm selects the robot r on the axis of C, with $N(r) = \emptyset$, and having smallest position in the *LSS*. The selected robot is unique. Hence, r is moved on one of its neighbors not belonging to the axis. This movement creates an asymmetric stationary configuration $C(1)$. When $C(1)$ is further elaborated, $\mathcal{A}_{break}$ terminates without computing any move. In fact, since $C(1)$ is asymmetric, all tests at Lines 6, 12, and 15 return false. Moreover, also tests at Lines Lines 2 and 8 return false otherwise, according to the move performed by r in $C(0)$, also $C(0)$ would have been in aRot$\cup$aDia. Notice that computing no move, the termination property is satisfied.

If in $C(0)$ the test at Line 12 is true, then $C \in$ fRef. According to the definition of fRef, C is reflective and hence there exist robots on the axis of C with free paths. Accordingly, $\mathcal{A}_{break}$ selects a unique robot r on the axis of C closest to $bp(R)$. Then, r moves along any free path so that to increase its distance from the center of $bp(R)$. Such a movement creates a configuration $C(1)$ which is still in fRef. When $C(1)$ is further elaborated, $\mathcal{A}_{break}$ correctly detects it as a configuration in fRef since all tests at Lines 2, 6, and 8 return false. In fact, since $C(1)$ is reflective, it can be neither in uRot nor in fRot; moreover, according to the movement of r, $C(1)$ can be neither a-rotational nor a-diagonal, otherwise $C(0)$ would have been too. Hence, in a finite number of movements of robots along the axis, a configuration belonging to uRef will be eventually produced. From there, as shown above, an asymmetric stationary configuration is created in just one step.

If in $C(0)$ the test at Line 6 is passed, then $C \in$ uRot. By definition, C is rotational and its central robot r has free paths or $N(r) = \emptyset$ holds. $\mathcal{A}_{break}$ moves the central robot r on one of its unoccupied neighbors. If r moves to a neighbor not belonging to an axis of reflection, the obtained configuration is stationary and asymmetric. Otherwise, a stationary reflective configuration $C(1)$ is created. When $C(1)$ is further elaborated, $\mathcal{A}_{break}$ detects it as a configuration in fRef because the path along which the robot r has moved defines the axis of reflection, and the robot at the end of the path have free paths. Moreover, in $C(1)$ tests at Lines 2 and 8 return false

since if $C(1)$ is detected as a-rotational or a-diagonal, also $C(0)$ would belong to the same class. Hence, $\mathcal{A}_{break}$ processes $C(1)$ at Line 12 and, as shown above, it produce a stationary asymmetric configuration in a finite time. □

The following two lemmata deal with the correctness of $\mathcal{A}_{break}$ with respect to input configurations belonging to aRot or aDia. Notice that in such cases the algorithm may produce pending moves.

Lemma 11: Let C be any initial configuration belonging to the class aRot. *Then, $\mathcal{A}_{break}$ is able to solve the SB problem with respect to C.*

Proof: Since C belongs to aRot, according to Definition 6 and Lemma 9, we get that there exists a unique robot r in C such that (1) r is blocked in C, (2) there exists a rotational-partition for $\mathcal{S}(r)$, and (3) $Reduce(C, r)$ is rotational of type 1 and r is central in $Compact(C, r)$. Notice that in C any type of symmetry may exist: C can be rotational, reflective (diagonal or not), or even asymmetric.

Let $\mathbb{E} : C = C(0), C(1), \ldots$ be any execution of $\mathcal{A}_{break}$ that starts from C. When $\mathcal{A}_{break}$ is executed with respect to $C(0)$, *MakeSpace* is called at line 5, and both r and a rotational-regular partition $\mathcal{P} = \{S_1, S_2 \ldots, S_k\}$ of $\mathcal{S}(r)$ are passed as input to that procedure. Two cases must be analyzed, according whether there exist a multiset $S_i \in \mathcal{P}$ having different strings or not.

- Assume that the strings in each set S_i belonging to $\mathcal{S}(r)$ are all the same.
  As a consequence, the block of code at lines 7–9 is executed. There, only the strings generated from r that starts with 1 are considered (they are generated from the hlines passing through an occupied neighbor of r). Let s be any of such strings: the robot r' corresponding to the 1 in the first occurrence of the substring 10 of s moves away from r along the hline corresponding to s. Notice that according to the definition of strings generated from r, the substring 10 always occurs in s.
  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 also in aRot, and in particular the same robot r that was central in $C(0)$ is now the robot that makes $C(1)$ a-rotational. This implies that when $\mathcal{A}_{break}$ processes $C(1)$, again *MakeSpace* is called at line 5, and both r and $\mathcal{S}(r)$ are passed as input to that procedure. Notice that $\mathcal{S}(r)$ in $C(1)$ may be different from the same multiset computed in $C(0)$ (it may happen that some of the previous S_i in the initial partition are now merged).
  Concerning $C(1)$, three cases may apply with respect to robots that had to move in $C(0)$: (*i*) they all moved and finished the move in $C(1)$, (*ii*) only a subset was activated, but all the activated robots finished the move in $C(1)$, and (*iii*) there are pending robots in $C(1)$. In cases (*i*) and (*ii*) the configuration is stationary, whereas in (*iii*) it is not.
  If $C(1)$ matches case (*i*), we are in the same situation as in $C(0)$. If case (*ii*) or (*iii*) applies, in the partition $\mathcal{P}$ received as input by *MakeSpace* there can be a mutiset S_i containing different strings. In such a case, Lines 2–5 are executed. Notice that now the procedure moves exactly those robots in S_i that was not activated in $C(0)$ or pending in $C(1)$. This implies that, according to the fairness property, all such robots are moved so that all the strings in S_i are the same as $\max(S_i)$ in $C(1)$ and all the robots related to these strings become stationary. According to this analysis, it follows that through repeated calls to procedure *MakeSpace*, the algorithm pushes the robots forward until the central robot r in $C(0)$ becomes unblocked in an obtained configuration $C(t)$, with $t > 1$. Notice that t is finite since it simply depends on Γ, that is the length of the longest string in $\mathcal{S}(r)$ computed in $C(0)$. It can be observed that $C(t)$ is stationary, initial, and belongs to uRot. According to Lemma 10, $\mathcal{A}_{break}$ eventually solves the *SB* problem after starting to process $C(t)$.
- Assume there exists a multiset $S_i \in \mathcal{P}$ containing different strings. In this case we have the same analysis performed for $C(1)$ in the previous case. In particular, we can assume that case (*ii*) above applies, and in $C(1)$ procedure *MakeSpace* will move robots that potentially can make the strings in $\mathcal{S}(r)$ all the same. Hence, as above, the execution proceeds until an asymmetric stationary configuration is created after a finite time.

□

Lemma 12: Let C be any initial configuration belonging to the class aDia. *Algorithm $\mathcal{A}_{break}$ is able to solve the SB problem with respect to C.*

Proof: If C is elaborated by $\mathcal{A}_{break}$ before Line 8, then according to Lemmata 10 and 11 it is correctly transformed into a stationary asymmetric configuration in a finite time.

Assume that C is elaborated by $\mathcal{A}_{break}$ at Line 8. Since C belongs to aDia, according to Definition 6 and Lemma 9 there exists a unique robot r in C such that (1) r is blocked in C, (2) there exists a reflective-partition for $\mathcal{S}(r)$, and (3) $Reduce(C, r)$ is diagonal and r is pivot in $Compact(C, r)$.

Let $\mathbb{E} : C = C(0), C(1), \ldots$ be any execution of $\mathcal{A}_{break}$ that starts from C. When $\mathcal{A}_{break}$ processes $C(0)$, *MakeSpace* is called at line 5, and both r and a reflective-partition $\mathcal{P} = \{S_1, S_2 \ldots, S_k\}$ of $\mathcal{S}(r)$ are passed as input to that procedure.

It can be observed that $C(1)$ still remains in the class aDia, but it could belongs also to other classes processed by $\mathcal{A}_{break}$. According to the order in which the algorithm checks the membership of the processes configuration, different cases may occur:

- $C(1)$ belongs to aRot. In that case, it will be processed by $\mathcal{A}_{break}$ at Line 2 and again *MakeSpace* is executed. The robot r passed as an argument to *MakeSpace* in $C(0)$ is still selected to be passed again to that procedure: this easily holds since r was the pivot robot in $Compact(C(0), r)$. Hence, *MakeSpace* continues to push forward the same robots involved in $C(0)$.

According to Lemma 11, from $C(1)$ the algorithm will eventually create a stationary asymmetric configuration.
- $C(1)$ belongs to uRot, but this case occurs when *MakeSpace* has made r unblocked and this implies that the configuration is stationary. According to Lemma 10, form $C(1)$ the algorithm solves the *SB* problem.
- $C(1)$ belongs to aDia. In this case, as already analyzed in the proof of Lemma 11, repeated calls to procedure *MakeSpace* lead the pivot robot r to make unblocked in a configuration $C(t)$, for a finite $t > 1$. $C(t)$ is stationary, initial, and belongs to uRef. According to Lemma 10, $\mathcal{A}_{break}$ eventually solves the *SB* problem after starting to process $C(t)$.

The above case analysis concludes the proof. □

Theorem 13: *$\mathcal{A}_{break}$ is able to solve the SB problem with respect to any initial configuration $C = (G, R, \mu)$ such that $G \in \{G_S, G_T\}$.*

Proof: It follows by using Lemmata 10, 11, and 12. □

VI. THE LINE FORMATION PROBLEM AS A CASE STUDY

The goal of the Line Formation problem (*LF*, for short) is to design a distributed algorithm $\mathcal{A}$ that, starting from any configuration $C = (G, R, \mu)$ such that $G \in \{G_S, G_T\}$, guides the robots to form a *line-pattern*. Given $n \geq 3$ robots, they form a line-pattern in G if all the robots lie on the same canonical line and form a path of n vertices (i.e., the robots are consecutive in the canonical line). Formally, an algorithm $\mathcal{A}$ solves the *LF* problem for any configuration C 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 robots in $C(t^*)$ form a line-pattern and no robot moves after t^*, i.e., $C(t) = C(t^*)$ holds for all $t \geq t^*$.

A. A NECESSARY CONDITION FOR SOLVING LF

In this section, we show that there are initial configurations in which the *LF* problem cannot be solved.

Definition 14: *Let $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$, be an initial configuration. We say that C is LF-*unresolvable *if one of the following cases apply*:
1) *C is reflective with no robot on the reflection axis;*
2) *C is rotational, $\rho(C) \neq 180°$, and the center of C is unoccupied;*
3) *C is rotational of type 1 or 3, $\rho(C) = 180°$, and the center of C is unoccupied.*

For sake of simplicity, in the remainder, we use $\mathcal{U}$ to denote the class containing all the "*LF*-unresolvable" configurations.

The following result motivates the term *LF*-unresolvable since it formally proves that the *LF* problem cannot be solved in each configuration belonging to $\mathcal{U}$.

Theorem 15: *Given an input configuration $C \in \mathcal{U}$, the LF problem cannot be solved in C, even for FSync robots.*

Proof: By contradiction, let us assume that there exists an algorithm $\mathcal{A}$ able to solve the *LF* problem for C. We analyze different cases according to the kind of symmetry holding in C.
- C is reflective with no robots on the reflection axis. This means that no robot can be moved from the axis to break the symmetry and also that all robots are partitioned into subsets $R_1, R_2, \ldots, R_k$, $k \geq 2$, with R_i formed exactly by two equivalent robots for each i. Let L be the line coincident with the axis of reflection. Assume that $\mathcal{A}$ wants to form the pattern on a canonical line parallel to the reflection axes. If that line does not coincide with the axis, then all robots in C should be moved in one of the half-planes defined by L. But the adversary may force each pair of robots in R_i to move symmetrically, and hence the two robots will be always in different half-planes. If $\mathcal{A}$ move the robots on L, then multiplicities will be formed on L. Notice that the adversary does not allow to break the multiplicity. Assume that $\mathcal{A}$ wants to form the pattern on a canonical line not parallel to the reflection axes. As above, the adversary will perform symmetric moves for the two robots in R_i. Even if $\mathcal{A}$ is able to make all robots on the same line L', at the intersection vertex between L and L' there are either no robots or two robots. In any case, $\mathcal{A}$ cannot form a line-pattern.
- C is rotational, $\rho(C) \neq 180°$, and the center of C is unoccupied. This means that all robots can be partitioned into subsets $R_1, R_2, \ldots, R_k$, $k \geq 1$, with robots in R_i pairwise equivalent and $|R_i| = n/k > 2$, for each i. Since the robots in any subset R_i are pairwise equivalent, for any move planned by $\mathcal{A}$, the adversary can force all the robots to move accordingly, and hence they always remain equivalent. This implies that robots in R_i cannot be guided by $\mathcal{A}$ to form any line-pattern.
- C is rotational, $\rho(C) = 180°$, and the center of C is unoccupied. This means that all robots can be partitioned into subsets $R_1, R_2, \ldots, R_k$, with $|R_i| = 2$ and the two robots in R_i are equivalent, for each i. As in the previous case, the adversary can force each pair of robots in R_i to perform symmetric moves, so they always maintain the same symmetry. The only way for $\mathcal{A}$ to form a line-pattern is to "rotate" all the pairs of robots with respect to the center of rotation so that they finally lie on the same canonical line.

  Consider the cases in which C is of type 1 or 3. In the first case, even if $\mathcal{A}$ is able to move all robots on the same canonical line, the obtained configuration does not form a line-pattern. This is implied by the fact that the initial configuration C does not have a robot on the center of rotation and the center of rotation must be part of the line-pattern. In the second case, being the center of the rotation not on a canonical line, then the line-pattern cannot be formed. □

At this point, in order to characterize the solvability of the *LF* problem, it emerges the problem of designing an algorithm for solving *LF* in any initial configuration C not belonging to $\mathcal{U}$. For sake of clarity, in the remainder we denote as $\mathcal{R}$ (shorthand for "resolvable") the class containing all such configurations. Formally:

Definition 16: $\mathcal{R}$ *is the class containing any initial configuration* C *fulfilling one of the following conditions*:

1) C *is asymmetric;*
2) C *is reflective, with robots on the reflection axis;*
3) C *is rotational of type 1, and the center of* C *is occupied;*
4) C *is rotational of type 2 with* $\rho(C) = 180°$ *and not reflective.*

It can be easily observed that classes $\mathcal{R}$ and $\mathcal{U}$ form a partition of all the initial configurations in $\mathcal{I}$.

In the remainder, we provide an algorithm called $\mathcal{A}_{LF}$ able to solve LF in any configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$, such that $C \in \mathcal{R}$. As a consequence, each configuration in $\mathcal{R}$ can be thought as a "LF-resolvable" configuration. Notice that $\mathcal{A}_{LF}$ is designed to take advantage of $\mathcal{A}_{break}$ as follows: $\mathcal{A}_{break}$ is responsible to manage leader configurations (exactly all the configurations referred to in Items 2 and 3 of Definition 16), while an algorithm denoted as $\mathcal{A}_{LF^-}$ handles all configurations referred to in Items 4 and 1 of the same definition. For the sake of presentation, algorithm $\mathcal{A}_0$ contains a module called $\mathcal{A}_0$ responsible for configurations in Item 4 only.

B. SOLVING LF IN ROTATIONAL CONFIGURATIONS

In this section, we focus on solving the LF problem on any configuration C as in Item 4 of Definition 16, that is C is rotational of type 2 with $\rho(C) = 180°$ and not reflective. To this end, we provide an algorithm denoted as $\mathcal{A}_0$. During its execution, and before generating the requested line-pattern, $\mathcal{A}_0$ may produce configurations as in the following definition.

Definition 17: Given any configuration $C = (G, R, \mu)$, *with* $G \in \{G_S, G_T\}$, *we say that* C *belongs to the class* $\mathcal{C}_0$ *if one of the following properties hold*:

1) C *is rotational of type 2 with* $\rho(C) = 180°$ *and not reflective;*
2) *there exists a unique pair of robots* (r', r'') *in* C *such that if* r' *or* r'' *is moved into an unoccupied neighboring vertex, then the move generates a configuration* C' *such that: (1)* C' *is rotational of type 2 with* $\rho(C') = 180°$ *and not reflective, and (2)* r' *and* r'' *are equivalent in* C'. *In this case,* r' *and* r'' *are called* companion *robots in* C.

Remark 18: Class $\mathcal{C}_0$ *can be partitioned into rotational and asymmetric configurations. In particular, the former corresponds to all the configurations fulfilling the first condition in Definition 17, and the latter to all the configurations fulfilling the second condition in Definition 17.*

According to this remark, in the following we will simply write "rotational configuration in $\mathcal{C}_0$" as a shorthand for any configuration fulfilling the first condition in Definition 17.

Let C be a rotational configuration with n robots belonging to $\mathcal{C}_0$. By definition, n is even and the center of C is the middle point of some edge e. We denote by $c(e)$ the center of C, by X the canonical line coincident with e. Denote also as Y' and Y'' the lines passing through $c(e)$ and parallel to the canonical directions different form X (cf. Figure 6).

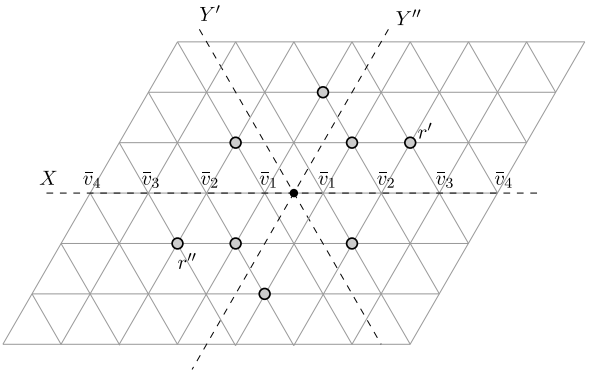

FIGURE 6. *A rotational configuration* $C \in \mathcal{C}_0$. *Here,* $q(\bar{C}') = (3, 11210)$ *and* $q(\bar{C}'') = (2, 12200)$. *Since* $q(\bar{C}') > q(\bar{C}'')$, *then* X *and* $Y \equiv Y'$ *can be used as two unoriented axes. In the obtained reference system, the equivalent robots* r' *and* r'' *are both in position* $(3, 1)$.

We now associate to C two auxiliary configurations defined as follows: $\bar{C}'$ ($\bar{C}''$, respectively) is the configuration obtained from C by projecting on X - along directions parallel to Y' (Y'', respectively) - all robots in C. Notice that both in $\bar{C}'$ and $\bar{C}''$ all robots are placed on X.

We now define some measures on both $\bar{C}'$ and $\bar{C}''$. Let X' and X'' the two half-lines starting from $c(e)$ and forming X. Denote as $\bar{v}_1, \bar{v}_2, \ldots$ the vertices in X' and X'' starting from $c(e)$, and proceeding in order (cf. Figure 6). Measures $q_1()$ and $q_2()$ are defined on $C^* \in \{\bar{C}', \bar{C}''\}$ as follows:

- $q_1(C^*) = t_1$, where t_1 is the size of the longest sequence $\bar{v_1}, \bar{v_2}, \ldots, \bar{v_{t_1}}$ starting from $\bar{v_1}$ and composed of occupied vertices; notice that $q_1(C^*) = 0$ when $\bar{v}_1$ is unoccupied;
- let t_2 is the largest index such that $\bar{v}_{t_2}$ is occupied and let $k = \max\{t_2, n/2\}$: $q_2(C^*)$ is equal to the integer expressed in base $b = n/2 + 1$ and composed of $k + 1$ digits such that the first digit is 1, and the i-th digit, $i > 1$, corresponds to $mul(\bar{v}_{i-1})$. By recalling that $mul()$ denotes the multiplicity of a vertex, and by observing that C rotational in $\mathcal{C}_0$ implies $mul(\bar{v}_i) < n/2$ for each $\bar{v}_i$ in X, then $q_2(C^*)$ can be formally defined as:

$$q_2(C^*) = b^k + \sum_{i=1}^{k} b^{k-i} \cdot mul(\bar{v}_i)$$

Define $q(C^*) = (q_1(C^*), q_2(C^*))$ as the final measure to be associated with C^*, and consider the pairs defining $q()$ as lexicographically ordered.

It can be easily observed that

$$M(C^*) = (n/2, \sum_{i=n/2}^{0} b^i)$$

is the maximum value for $q(C^*)$ and it occurs if and only if the n robots in C^* are uniformly distributed along the n vertices denoted as $\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_{n/2}$ and located on X (recall that, being X unordered, there are two occurrences of $\bar{v}_i$ in X for each $i = 1, 2, \ldots, n/2$, one in the half-line X' and the other in X'').

The following statement ensures that $q()$ can be always used to discriminate between lines Y' and Y''.

Lemma 19: If C is any rotational configuration in $\mathcal{C}_0$ then $q(\bar{C}') \neq q(\bar{C}'')$.

Proof: It can be easily observed that $q(\bar{C}') = q(\bar{C}'')$ only if both the following conditions hold: (1) all robots are located on the portion of the grid enclosed by Y' and Y'' and not containing the endpoints of the edge e defining the center of rotation for C, and (2) X is a reflection axis for C. Notice that the existence of a reflection axis for C contradicts $C \in \mathcal{C}_0$, since each configuration is such class is not reflective. □

Now, if $q(\bar{C}') > q(\bar{C}'')$ then define

- $Y \equiv Y'$ and $\bar{C} \equiv \bar{C}'$

else

- $Y \equiv Y''$ and $\bar{C} \equiv \bar{C}''$

All the notation and concepts defined so far will be used by $\mathcal{A}_0$ to solve the *LF* problem for any configuration C belonging to $\mathcal{C}_0$. In particular, the defined lines X and Y are used as unoriented axes forming a unique reference system computable to all robots (note that these axes are unoriented because C is symmetric), whereas $\bar{C}$ is used to recall which auxiliary configuration provided the larger value for $q()$.

The pseudo-code of $\mathcal{A}_0$ is given in Algorithm 3. The proof of the following lemma provides a description of the algorithm along with its correctness.

Algorithm 3 $\mathcal{A}_0$

Input: A configuration $C \in \mathcal{C}_0$.
Output: A configuration containing a line-pattern

1: **if** $q(\bar{C}) < M(\bar{C})$ **then**
2: **if** C is rotational **then**
3: let $t_1 = q_1(\bar{C})$
4: **if** in $\bar{C}$: $t_1 > 0$ and $\sum_{1 \leq j \leq t_1} mul(\bar{v}_j) > t_1$ **then**
5: in $\bar{C}$: let $i \leq t_1$ be the largest index such that $mul(\bar{v}_i) > 1$
6: let (i, j), with largest j, be the position of an occupied vertex
7: move the two equivalent robots in (i, j) to $(i + 1, j)$
8: **else**
9: in $\bar{C}$: let $i > t_1$ be the largest index such that $mul(\bar{v}_j) = 0$ for each $t_1 < j \leq i$
10: let $(i+1, j)$, with largest j, be the position of an occupied vertex
11: move the two equivalent robots in $(i + 1, j)$ to (i, j)
12: **else**
13: let r' and r'' be the unique pair of companion robots in C
14: move a robot between r' and r'' so that to obtain a rotational configuration $C^+ \in \mathcal{C}_0$ with $q(\bar{C}^+) > q(\bar{C})$
15: **else**
16: **if** C is rotational **then**
17: let i the minimal index such that there exist a robot in (i, j), $j > 0$
18: move the two equivalent robots in (i, j) to $(i, j - 1)$
19: **else**
20: let i the unique index such that there exist two robots in positions (i, j) and $(i, j + 1)$, for some $j > 0$
21: move the robot from $(i, j + 1)$ to (i, j)

Lemma 20: $\mathcal{A}_0$ solves LF in each configuration $C \in \mathcal{C}_0$.

Proof: The strategy of $\mathcal{A}_0$ is based on two different phases.

The first one (cf. Lines 2–14) aims to move robots along canonical lines parallel to X so that, at the end of the phase, for each $i = 1, 2, \ldots, n/2$ there exists a robot in a position with X-coordinate i in both the half-planes defined by Y. This is performed by either "pushing forward" robots so to increase $q_1(\bar{C})$ (cf. Lines 5–7) or "pulling back" robots so to increase $q_2(\bar{C})$ (cf. Lines 9–11). Notice that, in both cases, two equivalent robots are moved. If both robots move, the obtained configuration is still rotational in $\mathcal{C}_0$, otherwise an asymmetric configuration in $\mathcal{C}_0$ is generated. In the last case, moves at Lines 13 and 14 are performed in order to force the unmoved robot to complete the step and to create again a rotational configuration in $\mathcal{C}_0$. Notice the global reference system given by X and Y it is maintained throughout the execution of the phase since the moves always increase the value of $q(\bar{C})$.

In the second phase, $\mathcal{A}_0$ simply maintains each robot in its X-coordinate, but at each step it decreases the Y-coordinate of each moving robot (cf. Lines 17–18). Also in this second phase, it is considered the possibility that the adversary may activate only one robot: in such a case moves at Lines 20–21 force the unmoved robot to complete the required step. Notice that (1) $q(\bar{C})$ is not affected by such moves and hence Y will remain the same as in the first phase, and (2) as in the first case any obtained configuration is still in $\mathcal{C}_0$. At the end of the second phase a line-pattern will be formed on the X axis.

Since the number of moves required in each phase is bounded, it is clear that there will a finite time t^* such that the observed configuration $C(t^*)$ contains a line-pattern. Notice that, after t^*, $\mathcal{A}_0$ will no longer move any robot. □

The following additional result will be exploited later when $\mathcal{A}_0$ is used in conjunction with $\mathcal{A}_{break}$.

Lemma 21: For each configuration $C \in \mathcal{C}_0$ and for each possible execution $\mathbb{E} : C = C(0), C(1), \ldots$ of $\mathcal{A}_0$, any produced configuration $C(i)$, $i > 0$, does not belong to $\mathsf{aRot} \cup \mathsf{uRot} \cup \mathsf{aDia} \cup \mathsf{fRef} \cup \mathsf{uRef}$.

Proof: In the proof of Lemma 20, we have already remarked that $C(i) \in \mathcal{C}_0$. Hence, to prove the statement it is sufficient to show that $\mathcal{C}_0 \cap (\mathsf{aRot} \cup \mathsf{uRot} \cup \mathsf{aDia} \cup \mathsf{fRef} \cup \mathsf{uRef}) = \emptyset$.

Since each configuration in $\mathcal{C}_0$ is not reflective, then $\mathcal{C}_0 \cap (\mathsf{fRef} \cup \mathsf{uRef}) = \emptyset$ easily follows. Since each rotational configuration in $\mathcal{C}_0$ is of type 2, then $\mathcal{C}_0 \cap \mathsf{uRot} = \emptyset$ as well. In the remainder, let C' be any configuration in $\mathcal{C}_0$. We now show that $C' \notin \mathsf{aRot}$. If C' is rotational, being the center of C' of type 2, then clearly $C' \notin \mathsf{aRot}$. Consider the case in which C' is asymmetric and let $C'' = C' \setminus \{r', r''\}$ where r' and r'' are companion robots in C'. By definition, C'' is rotational of type 2 and belongs to $\mathcal{C}_0$. Assume that the center of C'' is the median point $c(e)$ of an edge $e = (v', v'')$. We recall that if $C' \in \mathsf{aRot}$ then there must exist a robot r which is central in $Compact(C', r)$. The only possibility for such a central robot r is that r is located on a vertex $v \in \{v', v''\}$. Of course, it is impossible that at the same time the following

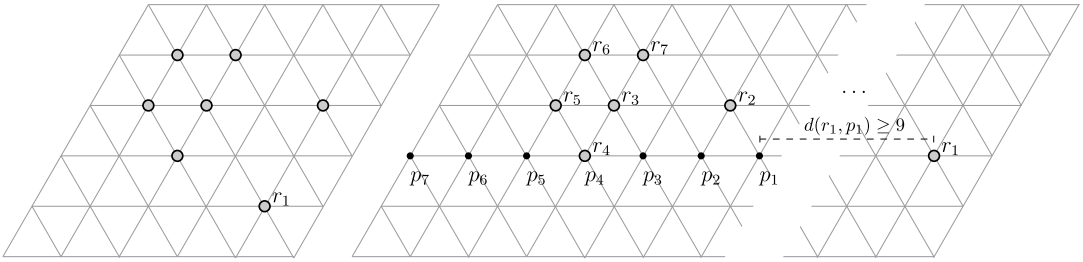

FIGURE 7. ***(left)*** **An input configuration C for the *LF* problem.** ***(right)*** **The configuration C' obtained by $\mathcal{A}_{LF^-}$ at the end of the first task (Reference System).**

two conditions hold: (1) C'' is rotational and its center is $c(e)$ with $e = (v', v'')$, and (2) C'' along with the companion robots r' and r'' is almost-rotational, there exists a robot r located on $v \in \{v', v''\}$ such that r is central in $Compact(C', r)$. Similar arguments can also be used to show that if $C' \in C_0$ then $C' \notin$ aDia. □

C. SOLVING LF IN ROTATIONAL OR ASYMMETRIC CONFIGURATIONS

In the previous section, we have shown that $\mathcal{A}_0$ solves *LF* on $\mathcal{C}_0$, a subset of all the *LF*-resolvable configurations in $\mathcal{R}$. Here we extend this result by providing an algorithm $\mathcal{A}_{LF^-}$ able to solve *LF* for each configuration which is in $\mathcal{C}_0$ or is asymmetric. Notice that $\mathcal{A}_{LF^-}$ incorporates $\mathcal{A}_0$.

The provided algorithm considers a partition of all the initial asymmetric configurations into three classes denoted as $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$. The algorithm exploits these classes so that each class is associated with a specific task for the robots. Informally, in any configuration in $\mathcal{C}_1$, robots have to cooperate in order to define a common reference system; in $\mathcal{C}_2$, robots exploit the formed reference system to partially form the requested line-pattern, and finally, in $\mathcal{C}_3$ robots complete the line-pattern formation. This approach follows the methodology proposed in [16]. More details are provided in the following paragraphs.

1) CLASS $\mathcal{C}_1$

This class concerns all the asymmetric configurations in which it is necessary to form a common reference system that allows robots to uniquely identify the targets (the final destination vertices where the line-pattern will be formed). In general, the requested common reference system is obtained by moving some (minimal number of) robots into specific positions such that they can be used by any other robot as a reference. These robots are called *guards*. The realized reference system should imply a unique mapping from robots to targets, and this mapping should be maintained along all the movements of non-guard robots.

In $\mathcal{A}_{LF^-}$, a single robot denoted as r_1 is used as a guard. It is selected as the robot that maximizes the sum of the distances from all other robots (in case of ties, $LSS(R)$ is used to make r_1 unique and detectable by all robots) - cf Fig. 7. This guard is moved far away from all the other robots so that the obtained configuration has a number of useful properties: (1) there exists a unique canonical line U passing through r_1 and each $bp(R')$, where $R' = R \setminus \{r_1\}$, (2) all the robots in R' are in the same half-plane with respect to U, and (3) by "projecting" all the positions of robots in R' onto U, the distance between r_1 and the projection vertex onto U closest to r_1 is large enough to guarantee that during the next task r_1 will remain detectable and not equivalent to any other robot. In the subsequent task (i.e., partial line-formation), robots in R' will be moved onto U in order to realize the requested pattern. Fig. 7 (right side) shows the configuration obtained by $\mathcal{A}_{LF^-}$ at the end of this task: notice that once r_1 is placed, each robot can determine the vertices $p_1, p_2, \ldots, p_7$ where the requested pattern will be finalized.

2) CLASS $\mathcal{C}_2$

This class contains all the configurations in which the guard r_1 has been placed and hence part of the requested line-pattern can be realized. Thanks to the guard, all robots can agree on embedding robot in R' onto U starting from vertex p_1 and continuing with the adjacent vertices $p_2, p_3, \ldots, p_n$. Robot in R' are moved one at a time and in a given order: if p_i is the first unoccupied vertex on U, the robot closest to p_i, and with the smallest position in $LSS(R)$ in case of ties, moves toward p_i along any shortest path. To avoid forming a symmetric configuration before completing the pattern, the last robot in R' stops at distance one from its target. In this way, during this task, no undesired collisions are created, and only stationary and asymmetric configurations are generated. In Fig. 7, robots in R' are named from r_2 to r_7 according to the ordering in which they are moved during this task.

3) CLASS $\mathcal{C}_3$

This class contains all the configurations in which only two robots remain to be moved in order to finalize the requested line-pattern. In our strategy, it concerns moving guard r_1 and the robot (say r_n) that is at distance one from its target (the last robot moved in the previous task). Our algorithm first moves r_1 along the canonical line U till reaching vertex p_1, and then r_n.

4) FORMALIZING $\mathcal{C}_1$, $\mathcal{C}_2$, AND $\mathcal{C}_3$

In this paragraph, we first introduce some notation and then we use it to formalize the three above classes. Let $D()$ be

the function that returns the sum of distances of a given robot from any other robot, that is $D(r) = \sum_{r' \in R \setminus \{r\}} d(r, r')$. Given an asymmetric configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$, let $r_1 = \text{argmax}_{r \in R}\{D(r)\}$ and with the smallest position in $LSS(R)$ in case of ties, $R' = R \setminus \{r_1\}$, and $\Delta = \max\{n, w\}$, where w is the largest width of any $bp(R')$'s. Notice that r_1 is unique and that both R' and Δ depend on r_1. According to this notation, we define two Boolean variables P_2 and P_3 defined on both C and r_1.

Definition 22 (Variable P_2): *Given any configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$, we say that variable P_2 holds in C if all the following properties hold*:

- $P_{2.1}$ *there exists a unique canonical line U passing through r_1 and each $bp(R')$;*
- $P_{2.2}$ *all the robots in R' are in the same half-plane with respect to U (U included);*
- $P_{2.3}$ *given $Proj(R') = \{p_1, p_2, \ldots, p_s\}$, $s \leq n$, be the set containing all the vertices obtained by "projecting" all the positions of robots in R' onto U along canonical lines forming a canonical angle with U in the half-plane not containing robots, then $d(r_1, p_1) \geq 3\Delta$ (we assume that $d(r_1, p_i) < d(r_1, p_j)$ if $i < j$);*
- $P_{2.4}$ *given $U_R = \{r \in R : r \text{ is on } U\}$, then $|U_R| \leq n - 1$; if $|U_R| = n - 1$ then the robot not on U is at distance at least 2 from U.*

Definition 23 (Variable P_3): *Given any configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$, we say that variable P_3 holds in C if both the following properties hold*:

- $P_{3.1}$ *there exists a path $\pi = (p_1, p_2, \ldots, p_n)$ such that (1) π is coincident with a canonical line U, (2) p_1 is the vertex in π closest to r_1, (3) vertices $p_2, p_3, \ldots, p_{n-1}$ are all occupied, and (4) there is a robot r_n at distance 1 from p_n;*
- $P_{3.2}$ *r_1 is located on U.*

These variables can now be used to formally define the three classes of configurations:

- $\mathcal{C}_3$ be the class of configurations where P_3 holds;
- $\mathcal{C}_2$ be the class of configurations where P_2 holds;
- $\mathcal{C}_1$ be the class containing all the initial asymmetric configurations in $\mathcal{R} \setminus (\mathcal{C}_0 \cup \mathcal{C}_2 \cup \mathcal{C}_3)$.

It is worth observing that each robot, by using the data acquired during the `Look` phase in any `LCM` cycle, can evaluate variables P_2 and P_3, and in turn it can determine the membership of the observed configuration to $\mathcal{C}_2$ or $\mathcal{C}_3$. Moreover, by also using Definition 17, it can also determine the membership to $\mathcal{C}_1$.

Algorithm $\mathcal{A}_{LF^-}$ is formalized in Fig. 4. Basically, it determines which class among $\mathcal{C}_0, \ldots, \mathcal{C}_3$ the input configuration belongs to, and move robots accordingly.

5) CORRECTNESS OF $\mathcal{A}_{LF^-}$

To prove the correctness of $\mathcal{A}_{LF^-}$, we first analyze its behavior when it takes as input a configuration in $\mathcal{C}_0$, and then we prove that classes $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ form a partition of all the asymmetric configurations in $\mathcal{R} \setminus \mathcal{C}_0$.

Algorithm 4 $\mathcal{A}_{LF^-}$

Input: A configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$, belonging to $\mathcal{C}_0$ or asymmetric.
Output: A configuration containing a line-pattern.

1: **if** $C \in \mathcal{C}_0$ **then**
2: call $\mathcal{A}_0$
3: **if** $C \in \mathcal{C}_1$ **then**
4: r_1 moves toward any closest point t such that when t is reached the obtained configuration belongs to $\mathcal{C}_2$ - during the move, r_1 must increase its distance from each other robot
5: **if** $C \in \mathcal{C}_2$ **then**
6: let $Proj(R')$ and U as defined in variable P_2
7: let $p_i \in Proj(R')$ be the first unoccupied vertex on U: the robot closest to p_i, and with the smallest position in $LSS(R)$ in case of ties, moves toward p_i along any shortest path; the last robot moving stops at distance 1 from its target
8: **if** $C \in \mathcal{C}_3$ **then**
9: let U be the canonical line as defined in Property $P_{3.1}$. If $d(r_1, p_1) > 0$ then r_1 moves along U so that it reduces the distance from p_1. If $d(r_1, p_1) = 0$ then r_n moves on p_n

Lemma 24: *$\mathcal{A}_{LF^-}$ solves LF in each configuration $C \in \mathcal{C}_0$.*

Proof: It simply follows from Lemma 20 and from the observation that $\mathcal{A}_0$ always generates configurations belonging to $\mathcal{C}_0$. Hence, when $\mathcal{A}_{LF^-}$ takes a configuration $C \in \mathcal{C}_0$ as input, it continuously calls $\mathcal{A}_0$ until the line-pattern is formed. □

Lemma 25: *Classes $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ form a partition of all the configurations in $\mathcal{R} \setminus \mathcal{C}_0$.*

Proof: $\mathcal{C}_1$ is disjoint with both $\mathcal{C}_2$ and $\mathcal{C}_3$ by definition. Concerning $\mathcal{C}_2$, it is disjoint with $\mathcal{C}_3$ since each configuration C cannot fulfill at the same time properties $P_{3.1}$ and $P_{2.4}$. Finally, observe that the definition of $\mathcal{C}_1$ guarantees that the three classes $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ contain each possible asymmetric configuration in $\mathcal{R} \setminus \mathcal{C}_0$. □

According to Lemma 25, in order to prove the correctness of $\mathcal{A}_{LF^-}$, we have to show that all the following properties hold in configurations belonging to the above three classes:

- Prop_1: any configuration produced by $\mathcal{A}_{LF^-}$ and belonging to $\mathcal{C}_1$, $\mathcal{C}_2$, or $\mathcal{C}_3$ is stationary and asymmetric;
- Prop_2: from any configuration in $\mathcal{C}_i$, $i = 1, 2, 3$, no class $\mathcal{C}_j$ with $j < i$ can be reached by $\mathcal{A}_{LF^-}$;
- Prop_3: from any configuration in $\mathcal{C}_i$, $i = 1, 2$, within a finite number of `LCM` cycles, a class $\mathcal{C}_j$, with $j > i$, is reached by $\mathcal{A}_{LF^-}$.

The following lemmas are responsible for checking these properties.

Lemma 26: *All three properties Prop_1, Prop_2, and Prop_3 are valid in class $\mathcal{C}_1$.*

Proof: Let C be any configuration in $\mathcal{C}_1$ processed by $\mathcal{A}_{LF^-}$. According to the definition of $\mathcal{C}_1$, robots in C do not form a line-pattern and neither P_2 nor P_3 holds in C.

We prove that Prop_1 is valid in $\mathcal{C}_1$. Algorithm $\mathcal{A}_{LF^-}$ moves only r_1, which is unique in C by definition. This implies that each configuration obtained after the move is stationary. According to the definition of r_1 and to the move ($r1$

moves away from the other robots), it is easy to observe that the obtained configurations remain asymmetric, since r_1 is the robot that maximizes $D()$ and the movement increases the value $D(r_1)$.

Concerning Prop_2, we have to prove that each configuration C' produced from C does not belong to $\mathcal{C}_0$. This can be observed by recalling that $C \notin \mathcal{C}_0$, that is C' is asymmetric, and that the movement of r_1 – which continuously increases $D(r_1)$ – prevents the formation of an asymmetric configuration as in Item 2 of Definition 17.

Prop_3 holds since as soon as r_1 reaches its target, the obtained configuration belongs to C_2 and this requires a finite number of LCM cycles (it depends on just the distance between the position of r_1 in C and the final target). □

Lemma 27: All three properties Prop_1, Prop_2, *and* Prop_3 *are valid in class* $\mathcal{C}_2$.

Proof: Let C be any configuration in $\mathcal{C}_2$ processed by $\mathcal{A}_{LF^-}$. According to the definition of $\mathcal{C}_2$, variable P_2 holds in C.

We prove that Prop_1 is valid in $\mathcal{C}_2$, that is we show that any configuration obtained from C is stationary and asymmetric. The stationary property follows from the fact that the algorithm moves one robot at a time, whereas the asymmetric property is guaranteed by the position of r_1 with respect all robots in R'. In particular, $d(r_1, p_1) \geq 3\Delta$ implies that r_1 cannot have equivalent robots, and hence if any obtained configuration is symmetric the only possibility is that there is an axis of reflection coincident with U, but this cannot happens by property $P_{2.4}$.

Concerning Prop_2, it can be observed that for any obtained configuration C', C' is asymmetric and it still belongs to $\mathcal{C}_2$ since all properties defining P_2 are not affected by the move until the last robot in R' reaches U, but in that case, the obtained configuration belongs to $\mathcal{C}_3$. C' cannot belong to C_1 since it is disjoint with both $\mathcal{C}_2$ and $\mathcal{C}_3$. Finally, C' cannot belong to $\mathcal{C}_0$ since the position of r_1 prevents the formation of an asymmetric configuration as in Item 2 of Definition 17.

Prop_3 is valid in $\mathcal{C}_2$ since each time a robot moves, it reduces its distance from U, and hence in a finite number of cycles, a configuration in $\mathcal{C}_3$ is generated. □

The validity of Prop_1 and Prop_2 in C_3 is shown in the proof of the following lemma.

Lemma 28: $\mathcal{A}_{LF^-}$ *is able to solve LF in any configuration in* $\mathcal{C}_3$.

Proof: Let C be any configuration in $\mathcal{C}_3$ processed by $\mathcal{A}_{LF^-}$. According to the definition of $\mathcal{C}_3$, variable P_3 holds in C. This means that in C there exists a path $\pi = (p_1, p_2, \ldots, p_n)$ such that (1) π is coincident with a canonical line U, (2) p_1 is the vertex in π closest to r_1, (3) vertices $p_2, p_3, \ldots, p_{n-1}$ are all occupied, and (4) there is a robot r_n at distance 1 from p_n.

Prop_1 is valid in $\mathcal{C}_3$ because, as long as robot r_1 moves along U, the configurations obtained are all stationary and asymmetric (the latter due to the position of r_n).

Concerning Prop_2, observe that the movement of r_1 does not affect variable P_3. This implies that if $C \in \mathcal{C}_3$, then any configuration obtained during the movement of r_1 it remains in such a class. As soon as r_n moves on π, the final configuration forming the requested line-pattern is obtained. □

Lemma 29: Let $C = (G, R, \mu)$, *with* $G \in \{G_S, G_T\}$, *be an initial configuration. If* $C \in \mathcal{C}_0$ *or* C *is asymmetric, then* $\mathcal{A}_{LF^-}$ *is able to solve the LF problem in* C.

Proof: According to Lemma 24, the statement holds when $C \in \mathcal{C}_0$. In the remainder, assume that $C \in C_i$, $i = 1, 2, 3$.

Assume that C is provided as input to $\mathcal{A}_{LF^-}$. According to Lemma 25, there exists a single class (say $\mathcal{C}_i$) that contains C. According to Prop_1, any obtained configuration is stationary and asymmetric.

Now, let C' be any configuration generated from C. By Prop_2 and Prop_3, we can consider C' belonging to some class $\mathcal{C}_j$ with $j \geq i$. According to this analysis, we can say that C' will further evolve during the time by changing its membership from class to class according to the forward transitions defined by Lemmas 26 and-27. Although the execution of $\mathcal{A}_{LF^-}$ is infinite, property Prop_3 assures that the transition from each class into another is completed within a finite number of LCM cycles. This implies that a configuration in $\mathcal{C}_3$ will be generated. Then, according to Lemma 28, in a finite time $\mathcal{A}_{LF^-}$ will finally produce a configuration containing the requested line-pattern. □

Lemma 30: For each configuration C *which is in* $\mathcal{C}_0$ *or asymmetric and for each possible execution* $\mathbb{E} : C = C(0), C(1), C(2), \ldots$ *of* $\mathcal{A}_{LF^-}$, *any produced configuration* C_i, $i \geq 0$, *does not belong to* $\mathsf{aRot} \cup \mathsf{aDia} \cup \mathsf{uRot} \cup \mathsf{fRef} \cup \mathsf{uRef}$.

Proof: According to the proof of Lemma 20, we know that if $C(i) \in \mathcal{C}_0$ then $C(i+1)$ still belongs to $\mathcal{C}_0$. Consider now the cases in which $C(i) \in \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$. Since $C(i)$ is asymmetric (cf. Lemmas 26–28), then it cannot belong to $\mathsf{aRot} \cup \mathsf{fRef} \cup \mathsf{uRef}$.

Assume $C(i) \in \mathcal{C}_1$. Concerning the membership of $C(i+1)$ to $\mathsf{aRot} \cup \mathsf{aDia}$, notice that $C(i)$ belongs neither to aRot nor to aDia, because it is currently processed by $\mathcal{A}_{LF^-}$. Moreover, during the move, in order for the obtained configuration to be in one of the two classes aRot or aDia, there should be a robot r different from r_1 such that r_1 correspond to some bit 1 in some string belonging to $\mathcal{S}(r)$. But even if this happens, there can be no string in $\mathcal{S}(r)$ which is almost-equal to the one to which r_1 belongs. This property is ensured by the movement of r_1.

Assume $C(i) \in \mathcal{C}_2$. In the proof of Lemma 27 we have already observed that the position of r_1 guarantees that $C(i+1)$ is asymmetric. In particular, $d(r_1, p_1) \geq 3\Delta$ implies that r_1 cannot have equivalent robots, and hence if any obtained configuration is symmetric the only possibility is that there is an axis of reflection coincident with U, but this cannot happen by property $P_{2.4}$. For this reasons, we are sure that any obtained configuration cannot belong to $\mathsf{aDia} \cup \mathsf{aRot}$ (as in the proof of Lemma 26, there should be a robot r different from r_1 such that r_1 corresponds to some bit 1 in

some string belonging to $\mathcal{S}(r)$, but there can be no string in $\mathcal{S}(r)$ which is almost-equal to the one to which r_1 belongs).

Assume $C(i) \in \mathcal{C}_3$. The obtained configuration $C(i+1)$ cannot belong to $\mathsf{aDia} \cup \mathsf{aRot}$ because $Reduce(C(i+1), r)$ is neither rotational nor diagonal for each possible robot r (this simply derives from the $n-1$ robots located on U, whereas r_n is not on U but at distance 1 from it). □

D. SOLVING LF FOR ANY CONFIGURATION IN $\mathcal{R}$

Algorithm 5 shows $\mathcal{A}_{LF}$: it refers to how $\mathcal{A}_{break}$ can be customized by proving it (1) a module M responsible for managing the termination property concerning the *LF* problem, and (2) the algorithm $\mathcal{A}_{LF^-}$ responsible for solving *LF* for configurations which are asymmetric or belong to $\mathcal{C}_0$. In this form, $\mathcal{A}_{LF}$ is able to solve the *LF* problem for each configuration in $\mathcal{R}$.

Algorithm 5 $\mathcal{A}_{LF}$

Input: A configuration $C = (G, R, \mu) \in \mathcal{R}$, with $G \in \{G_S, G_T\}$, and composed of $n \geq 3$ ASYNC robots.
Output: A configuration containing a line-pattern.
1: let M be a simple procedure defined as follows: "*if robots in C form a line-pattern, then each robot performs the nil movement*"
2: call $\mathcal{A}_{break}$ and provide it with C, M, and $\mathcal{A}_{LF^-}$ as input

Theorem 31: The LF problem can be solved in any configuration $C = (G, R, \mu)$, with $G \in \{G_S, G_T\}$, if and only if $C \in \mathcal{R}$, that is either C is leader or C is rotational of type 2 with $\rho(C) = 180°$ and not reflective.

Proof: According to the adopted notation, all the initial configurations are partitioned into $\mathcal{U}$ and $\mathcal{R}$. Theorem 15 ensures that *LF* cannot be solved in any configuration in $\mathcal{U}$.

Assume that the input configuration $C \in \mathcal{R}$, when processed by $\mathcal{A}_{LF}$, is not elaborated by $\mathcal{I}_{Mod}$ nor $\mathcal{F}_{Mod}$. In such a case, C is a leader configuration and Theorem 13 ensures that the symmetry breaking problem is solved in C and hence, in a finite time, C is transformed into an asymmetric configuration C'. When C' is processed again by $\mathcal{A}_{LF}$, then either $\mathcal{I}_{Mod}$ or $\mathcal{F}_{Mod} \equiv \mathcal{A}_{LF^-}$ are called by $\mathcal{A}_{break}$. In the first case, *LF* will be certified by $\mathcal{I}_{Mod}$ to be solved; in the second case, Lemmata 29 and 30 ensure that $\mathcal{A}_{LF^-}$ will correctly solve *LF*.

Assume that the input configuration $C \in \mathcal{R}$, when processed by $\mathcal{A}_{LF}$, is elaborated by $\mathcal{I}_{Mod}$ or $\mathcal{F}_{Mod}$. If the *nil* move is computed by $\mathcal{I}_{Mod}$, then C contains already the requested line-pattern. If C is elaborated by $\mathcal{F}_{Mod}$ then it is either asymmetric or it belongs to $\mathcal{C}_0$. In both cases, as remarked above, Lemmata 29 and 30 ensure that $\mathcal{A}_{LF^-}$ will correctly solve *LF*. □

VII. CONCLUSION

This paper investigated the *Symmetry Breaking* problem in grid graphs by means of very weak robots. 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 the proposed algorithm $\mathcal{A}_{break}$ can solve the problem on both the square and triangular grids. The algorithm is proposed so that it can be also combined with other modules. As a case study, we used the proposed algorithm to fully characterize the *Line Formation* problem on the considered grids.

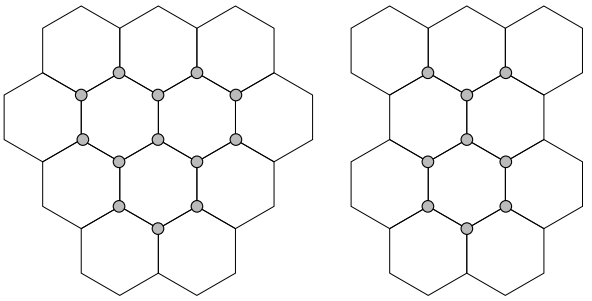

FIGURE 8. Example of rotational and reflective initial configurations defined on G_H.

This work opens some interesting investigation directions. The most obvious open problem is extending the proposed algorithm $\mathcal{A}_{break}$ to work in the case of hexagonal grids. In this case, the challenge would be to identify a single strategy valid for all the three types of graphs. As can be observed from Fig. 8, in the case of G_H it is not possible to "make space" around the central node (or around a node on the symmetry axis) by moving robots along hlines.

As another possible investigation, it would worth testing whether it is possible to combine the proposed algorithm with that proposed in [8] to solve the *Arbitrary Pattern Formation* problem. The combination, if realizable, could let to consider as input not only asymmetric but also leader configurations. This would lead to a complete characterization of the *Arbitrary Pattern Formation* problem on square and triangular grids. Finally, it would be interesting to analyze other problems in grid graphs that might benefit from $\mathcal{A}_{break}$.

ACKNOWLEDGMENT

An earlier version of this paper was presented in part at the 21st Italian Conference on Theoretical Computer Science (ICTCS), in 2020 [7].

REFERENCES

[1] R. Adhikary, K. Bose, M. K. Kundu, and B. Sau, "Mutual visibility by asynchronous robots on infinite grid," in *Proc. Int. Symp. Algorithms Exp. Sensor Syst., Wireless Netw. Distrib. Robot.* in Lecture Notes in Computer Science, vol. 11410. Cham, Switzerland: Springer, 2018, pp. 83–101.
[2] H. M. Barbera, J. P. C. Quinonero, M. A. Z. Izquierdo, and A. G. Skarmeta, "I-fork: A flexible AGV system using topological and grid maps," in *Proc. IEEE Int. Conf. Robot. Autom.*, Sep. 2003, pp. 2147–2152.
[3] S. Bhagat, S. G. Chaudhuri, and K. Mukhopadhyaya, "Formation of general position by asynchronous mobile robots under one-axis agreement," in *Proc. 10th Int. WS Algorithms Comput. (WALCOM)*, in Lecture Notes in Computer Science, vol. 9627. Kathmandu, Nepal: Springer, Mar. 2016, pp. 80–91.
[4] K. Bose, R. Adhikary, M. K. Kundu, and B. Sau, "Arbitrary pattern formation on infinite grid by asynchronous oblivious robots," *Theor. Comput. Sci.*, vol. 815, pp. 213–227, May 2020.
[5] K. Bose, M. K. Kundu, R. Adhikary, and B. Sau, "Optimal gathering by asynchronous oblivious robots in hypercubes," in *Proc. 20th Int. Symp. Algorithms Exp. Sensor Syst., Wireless Netw. Distrib. Robot. (Algosensors)* in Lecture Notes in Computer Science, vol. 11410. Cham, Switzerland: Springer, 2019, pp. 102–117.

[6] Q. Bramas and S. Tixeuil, "Arbitrary pattern formation with four robots," in *Proc. 20th Int. Symp. Stabilization, Saf., Secur. Distrib. Syst. (SSS)* in Lecture Notes in Computer Science, vol. 11201. Cham, Switzerland: Springer, 2018, pp. 333–348.

[7] S. Cicerone, "Breaking symmetries on tessellation graphs via asynchronous robots," in *Proc. 21st Italian Conf. Theor. Comput. Sci. (ICTCS)*, vol. 2756, G. Cordasco, L. Gargano, and A. Rescigno, Eds. CEUR-WS.org, 2020, pp. 122–136. [Online]. Available: http://ceur-ws.org/Vol-2756

[8] S. Cicerone, A. Di Fonso, G. Di Stefano, and A. Navarra, "Arbitrary pattern formation on infinite regular tessellation graphs," in *Proc. Int. Conf. Distrib. Comput. Netw.*, Jan. 2021, pp. 56–65.

[9] S. Cicerone, G. Di Stefano, and A. Navarra, "Gathering of robots on meeting-points: Feasibility and optimal resolution algorithms," *Distrib. Comput.*, vol. 31, no. 1, pp. 1–50, Feb. 2018.

[10] S. Cicerone, G. Di Stefano, and A. Navarra, "Asynchronous arbitrary pattern formation: The effects of a rigorous approach," *Distrib. Comput.*, vol. 32, no. 2, pp. 91–132, Apr. 2019.

[11] S. Cicerone, G. Di Stefano, and A. Navarra, "Asynchronous robots on graphs: Gathering," in *Distributed Computing by Mobile Entities: Current Research in Moving and Computing* (Lecture Notes in Computer Science), vol. 11340, P. Flocchini, G. Prencipe, and N. Santoro, Eds. Cham, Switzerland: Springer, 2019, pp. 184–217.

[12] S. Cicerone, G. Di Stefano, and A. Navarra, "Embedded pattern formation by asynchronous robots without chirality," *Distrib. Comput.*, vol. 32, no. 4, pp. 291–315, Aug. 2019.

[13] S. Cicerone, G. Di Stefano, and A. Navarra, "On gathering of semi-synchronous robots in graphs," in *Proc. 21st Int. Symp. Stabilization, Saf., Secur. Distrib. Syst. (SSS)*, in Lecture Notes in Computer Science, vol. 11914. Pisa, Italy: Springer, Oct. 2019, pp. 84–98.

[14] S. Cicerone, G. Di Stefano, and A. Navarra, "Gathering robots in graphs: The central role of synchronicity," *Theor. Comput. Sci.*, vol. 849, pp. 99–120, Jan. 2021.

[15] S. Cicerone, G. D. Stefano, and A. Navarra, "'Semi-asynchronous': A new scheduler in distributed computing," *IEEE Access*, vol. 9, pp. 41540–41557, 2021.

[16] S. Cicerone, G. Di Stefano, and A. Navarra, "A structured methodology for designing distributed algorithms for mobile entities," *Inf. Sci.*, vol. 574, pp. 111–132, Oct. 2021.

[17] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro, "Distributed computing by mobile robots: Gathering," *SIAM J. Comput.*, vol. 41, no. 4, pp. 829–879, Jan. 2012.

[18] G. D'Angelo, A. Navarra, and N. Nisse, "A unified approach for gathering and exclusive searching on rings under weak assumptions," *Distrib. Comput.*, vol. 30, no. 1, pp. 17–48, Feb. 2017.

[19] S. Das, P. Flocchini, N. Santoro, and M. Yamashita, "Forming sequences of geometric patterns with oblivious mobile robots," *Distrib. Comput.*, vol. 28, no. 2, pp. 131–145, Apr. 2015.

[20] G. Di Stefano and A. Navarra, "Gathering of oblivious robots on infinite grids with minimum traveled distance," *Inf. Comput.*, vol. 254, pp. 377–391, Jun. 2017.

[21] G. Di Stefano and A. Navarra, "Optimal gathering of oblivious robots in anonymous graphs and its application on trees and rings," *Distrib. Comput.*, vol. 30, no. 2, pp. 75–86, Apr. 2017.

[22] Y. Dieudonné, F. Petit, and V. Villain, "Leader election problem versus pattern formation problem," in *Proc. 24th Int. Symp. Distrib. Comput. (DISC)*, in Lecture Notes in Computer Science, vol. 6343. Springer, 2010, pp. 267–281.

[23] P. Flocchini, "Gathering," in *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, vol. 11340, P. Flocchini, G. Prencipe, and N. Santoro, Eds. Cham, Switzerland: Springer, 2019, pp. 63–82.

[24] P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta, "Distributed computing by mobile robots: Uniform circle formation," *Distrib. Comput.*, vol. 30, no. 6, pp. 413–457, Dec. 2017.

[25] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, "Arbitrary pattern formation by asynchronous, anonymous, oblivious robots," *Theor. Comput. Sci.*, vol. 407, nos. 1–3, pp. 412–447, Nov. 2008.

[26] P. Flocchini, G. Prencipe, and N. Santoro, Eds., *Distributed Computing by Mobile Entities, Current Research in Moving and Computing* (Lecture Notes in Computer Science), vol. 11340. Cham, Switzerland: Springer, 2019.

[27] N. Fujinaga, Y. Yamauchi, H. Ono, S. Kijima, and M. Yamashita, "Pattern formation by oblivious asynchronous mobile robots," *SIAM J. Comput.*, vol. 44, no. 3, pp. 740–785, Jan. 2015.

[28] S. Ghike and K. Mukhopadhyaya, "A distributed algorithm for pattern formation by autonomous robots, with no agreement on coordinate compass," in *Proc. 6th Int. Conf. Distrib. Comput. Internet Technol., (ICDCIT)*, in Lecture Notes in Computer Science, vol. 5966. Bhubaneswar, India: Springer, Feb. 2010, pp. 157–169.

[29] B. Grünbaum and G. C. Shepard, *Tiling and Patterns*. New York, NY, USA: W. H. Freeman & Co., 1987.

[30] S. Guilbault and A. Pelc, "Gathering asynchronous oblivious agents with local vision in regular bipartite graphs," *Theor. Comput. Sci.*, vol. 509, pp. 86–96, Oct. 2013.

[31] J. E. Ionascu, "Half domination arrangements in regular and semi-regular tessellation type graphs," 2012, *math > arXiv:1201.4624*.

[32] R. Klasing, E. Markou, and A. Pelc, "Gathering asynchronous oblivious mobile robots in a ring," *Theor. Comput. Sci.*, vol. 390, no. 1, pp. 27–39, Jan. 2008.

[33] G. Sharma, A. Dutta, and J. Kim, "Optimal online coverage path planning with energy constraints," in *Proc. AAMAS*, 2019, pp. 1189–1197.

[34] I. Suzuki and M. Yamashita, "Distributed anonymous mobile robots: Formation of geometric patterns," *SIAM J. Comput.*, vol. 28, no. 4, pp. 1347–1363, Jan. 1999.

[35] M. Yamashita and I. Suzuki, "Characterizing geometric patterns formable by oblivious anonymous mobile robots," *Theor. Comput. Sci.*, vol. 411, nos. 26–28, pp. 2433–2453, Jun. 2010.

[36] Y. Yamauchi, "Symmetry of anonymous robots," in *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, vol. 11340, P. Flocchini, G. Prencipe, and N. Santoro, Eds. Cham, Switzerland: Springer, 2019, pp. 109–133.

[37] Y. Yamauchi, T. Uehara, S. Kijima, and M. Yamashita, "Plane formation by synchronous mobile robots in the three dimensional Euclidean space," in *Proc. 29th Int. Symp. Distrib. Comput. (DISC)*, vol. 9363. Berlin, Germany: Springer, 2015, pp. 92–106.

SERAFINO CICERONE received the Ph.D. degree in computer science from the University of Rome "La Sapienza," in 1998. He is currently an Associate Professor with the Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy. His research interests include the specification, design, verification, implementation of efficient algorithms, distributed algorithms, combinatorial optimization, algorithm engineering, algorithmic graph theory, spatial, and geometric data.