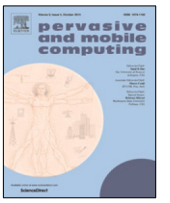
Pervasive and Mobile Computing 95 (2023) 101842

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

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



The geodesic mutual visibility problem: Oblivious robots on grids and trees☆

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

[a] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università degli Studi dell'Aquila, Via Veotio, I-67100, L'Aquila, Italy
[b] Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Via Vanvitelli 1, I-06123, Perugia, Italy

ARTICLE INFO

Article history:
Received 26 May 2023
Received in revised form 19 August 2023
Accepted 8 September 2023
Available online 11 September 2023

Keywords:
Autonomous mobile robots
Oblivious robots
Mutual visibility
Grids
Trees

ABSTRACT

The Mutual Visibility is a well-known problem in the context of mobile robots. For a set of n robots disposed in the Euclidean plane, it asks for moving the robots without collisions so as to achieve a placement ensuring that no three robots are collinear. For robots moving on graphs, we consider the Geodesic Mutual Visibility (GMV) problem. Robots move along the edges of the graph, without collisions, so as to occupy some vertices that guarantee they become pairwise geodesic mutually visible. This means that there is a shortest path (i.e., a "geodesic") between each pair of robots along which no other robots reside. We study this problem in the context of trees and (finite or infinite) square grids, for robots operating under the standard Look–Compute–Move model. In both scenarios, we provide resolution algorithms along with formal correctness proofs, highlighting the most relevant peculiarities arising within the different contexts, while optimizing the time complexity.

1. Introduction

We consider swarm robotics concerning autonomous, identical and homogeneous robots operating in cyclic operations. Robots are equipped with sensors and motion actuators and operate in standard *Look–Compute–Move* cycles (see, e.g., [3–5]). When activated, in one cycle a robot takes a snapshot of the current global configuration (`Look`) in terms of relative robots' positions, according to its own local coordinate system. Successively, in the `Compute` phase, it decides whether to move toward a specific direction or not and in the positive case it moves (`Move`). A Look–Compute–Move cycle forms a computational cycle of a robot. What is computable by such entities has been the object of extensive research within distributed computing, see, e.g., [4,6–12].

One of the basic tasks for mobile robots, intended as points in the plane, is certainly the requirement to achieve a placement so as no three of them are collinear. Furthermore, during the whole process, no two robots must occupy the

☆ Preliminary results appear in the proceedings of the 24th International Conference on Distributed Computing and Networking (ICDCN) 2023 (Cicerone et al., 2023, [1]), and in the proceedings of the 25th International Symposium on Stabilization, Safety, and Security of Distributed System (SSS) 2023 (Cicerone et al., 2023, [2]).

The work has been supported in part by project "SICURA - Casa intelligente delle tecnologie per la sicurezza", CUP C19C200005200004 - Piano di investimenti per la diffusione della banda ultra larga FSC, Italy 2014–2020 and by the Italian National Group for Scientific Computation, Italy (GNCS-INdAM).

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

https://doi.org/10.1016/j.pmcj.2023.101842

same position concurrently, i.e., *collisions* must be avoided. This is known as the MUTUAL VISIBILITY problem. The idea is that, if three robots are collinear, the one in the middle may obstruct the reciprocal visibility of the other two.

Mutual Visibility has been largely investigated in recent years in many forms, subject to different assumptions. One main distinction within the Look–Compute–Move model concerns the level of synchronicity assumed among robots. Robots are assumed to be synchronous [13], i.e., they are always all active and perform each computational cycle within the same amount of time; semi-synchronous [14–17], i.e., robots are not always all active but all active robots always perform their computational cycle within a same amount of time, after which a new subset of robots can be activated; asynchronous [15,17–22], i.e., each robot can be activated at any time and the duration of its computational cycle is finite but unknown. Robots are generally endowed with visible lights of various colors useful to encode some information (to be maintained across different computational cycles and/or communicated to other robots), whereas in [16] robots are considered completely oblivious, i.e., without any memory about past events. Usually, robots are considered as points in the plane but in [23], where robots are considered "fat", i.e., occupying some space modeled as disks in the plane. Furthermore, instead of moving freely in the Euclidean plane, in [18,22] robots are constrained to move along the edges of a graph embedded in the plane and still the mutual visibility is defined according to the collinearity of the robots in the plane.

In this paper, we introduce the GEODESIC MUTUAL VISIBILITY problem (GMV, for short): starting from a configuration composed of robots located on distinct vertices of an arbitrary graph, within finite time the robots must reach, without collisions, a configuration where they all are in *geodesic mutual visibility*. Robots are in geodesic mutual visibility if they are pairwise mutually visible, and two robots on a graph are mutually visible if there is a shortest path (i.e., a "geodesic") between them along which no other robots reside. This new problem can be thought of as a possible counterpart to the MUTUAL VISIBILITY for robots moving in a discrete environment.

While this concept is interesting by itself, its study is motivated by the fact that robots, after reaching a GMV condition, e.g., can communicate in an efficient and "confidential" way, by exchanging messages through the vertices of the graph that do not pass through vertices occupied by other robots or can reach any other robot along a shortest path without collisions. Concerning the last motivation, in [24], it is studied the COMPLETE VISITABILITY problem of repositioning a given number of robots on the vertices of a graph so that each robot has a path to all others without visiting an intermediate vertex occupied by any other robot. In that work, the required paths are not shortest paths and the studied graphs are restricted to infinite square and hexagonal grids, both embedded in the Euclidean plane.

The geodesic mutual visibility has been investigated in [25] from a pure graph-theoretical point of view in order to understand how many robots, at most, can potentially be placed within a graph G in order to guarantee GMV. Such a number of robots has been denoted by $\mu(G)$. In a general graph G, it turns out to be NP-complete to compute $\mu(G)$, whereas it has been shown that there are exact formulas for special graph classes like paths, cycles, trees, block graphs, co-graphs, and grids [25–27]. For instance, within a path P, at most two robots can be placed, i.e., $\mu(P) = 2$, whereas for a ring R, $\mu(R) = 3$. In a finite square grid G of $N > 3$ rows and $M > 3$ columns, $\mu(G) = 2\min\{M, N\}$, whereas for a tree T, it has been proven that $\mu(T) = \ell(T)$, with $\ell(T)$ being the number of leaves of T.

1.1. Our results

After formally defining the problem of achieving GMV starting from a configuration of robots disposed on general graphs, we focus on two main topologies: Square Grids and Trees.

The relevance of studying grids is certainly motivated by their peculiarity in representing a discretization of the Euclidean plane. For trees, the interest has been motivated by the apparent simplicity of the topology which actually hides many challenges. In both scenarios, robots are assumed to have no explicit means of communication or memory of past events. In fact, we consider oblivious robots without lights. Hence, the movement of a robot does rely only on local computations on the basis of the snapshot acquired in the `Look` phase. Furthermore, in order to approach the problem, we make use of the methodology proposed in [3] that helps in formalizing the resolution algorithms as well as the related correctness proofs.

SQUARE GRIDS. We consider GMV for robots moving on square grids embedded in the plane. Actually, we first consider finite grids and then provide relevant intuitions for extending the results to infinite grids. Furthermore, our algorithm is subject to the requirement to obtain a placement of the robots so that the final minimum bounding rectangle enclosing all the robots is of minimum area (this area-constrained GMV problem is denoted as GMV_{area}). We provide time-optimal algorithms that are able to solve GMV_{area} in both finite and infinite grid graphs. These algorithms work for synchronous robots endowed with chirality (i.e., a common handedness).

TREES. Given a tree T with $\ell(T)$ leaves, we first consider the extreme case of $n = \ell(T)$ robots disposed on $\ell(T)$ different vertices of T and we look for a distributed algorithm that makes robots moving so as to achieve GMV without incurring in collisions. Depending on the tree, the solution to the GMV problem is not unique, but an algorithm that moves robots so as to occupy all the leaves of T always solves the problem. We design a deterministic algorithm, identical for all the robots, that solves initial configurations when considering the very weak setting of semi-synchronous (and hence holding also for synchronous) robots. For the initial configurations, where each vertex is occupied by at most one robot, we assume the absence of *critical* vertices. Intuitively, a vertex v is said to be critical if two or more robots must pass through v in

order to reach a leaf, hence potentially colliding in v. The formal definition of critical vertex will be given successively. We then provide the necessary modifications to the algorithm for solving the case with $n \leq \ell(T)$. We also measure the time complexity of our algorithm in terms of *epochs* – where an epoch is the time duration for all the robots to execute at least one complete Look–Compute–Move cycle since the end of the previous epoch. We compare the obtained result with a lower bound that leaves some gaps for improvements. Finally, we provide an extended discussion about the configurations admitting critical vertices as well as the asynchronous or synchronous settings. In fact, the difficulties arising in such contexts deserve deep investigation and attention. However, we provide challenging ideas, strategies and observations in order to stimulate future research.

1.2. Outline

The rest of the paper is organized as follows. Section 2 introduces the robot model we have adopted. Section 3 formalizes the GMV problem and revises a resolution methodology to approach problems within the Look–Compute–Move context. Section 4 deals with GMV_{area} on grids. It starts with some notation specific to the grid case, and then the resolution algorithm along with its correctness proof is intuitively and formally provided according to the methodology recalled in Section 3. The section terminates with an informal description of the extension of the algorithm to deal with infinite grids. Section 5 concerns GMV on trees. It starts with some notation specific to the tree case. In particular, the formal definition of critical vertex is provided. Then, the resolution algorithm for initial configurations without critical vertices is provided along with its correctness proof according to the methodology recalled in Section 3. Finally, the section terminates with challenging scenarios to highlight difficulties arising when critical vertices are admitted. Section 6 concludes the paper, posing interesting future research directions.

2. Robot model

Robots are modeled according to $\mathcal{OBLOT}$ (e.g., see [28] for a survey), one of the classical theoretical models for swarm robotics. In this model, robots are computational entities that can move in some environment (a graph in our case) and can be characterized according to a large spectrum of settings. Each setting is defined by specific choices among a range of possibilities, with respect to a fundamental component – time synchronization – as well as other important elements, like memory, orientation and mobility. We assume such settings at minimum as follows:

- *Anonymous*: no unique identifiers;
- *Autonomous*: no centralized control;
- *Dimensionless*: no occupancy constraints, no volume, modeled as entities located on vertices of a graph;
- *Oblivious*: no memory of past events;
- *Homogeneous*: they all execute the same *deterministic*[1] algorithm;
- *Silent*: no means of direct communication;
- *Disoriented*: no common coordinate system.

Each robot in the system has sensory capabilities allowing it to determine the location of other robots in the graph, relative to its own location. Each robot refers in fact to a *Local Coordinate System* (LCS) that might be different from robot to robot. Each robot follows an identical algorithm that is pre-programmed into the robot. The behavior of each robot can be described according to the sequence of four states: `Wait`, `Look`, `Compute`, and `Move`. Such states form a computational cycle (or briefly a cycle) of a robot.

1. `Wait`. The robot is idle. A robot cannot stay indefinitely idle;
2. `Look`. The robot observes the environment by activating its sensors which will return a snapshot of the positions of all other robots with respect to its own LCS. Each robot is viewed as a point;
3. `Compute`. The robot performs a local computation according to a deterministic algorithm $\mathcal{A}$ (we also say that the robot executes $\mathcal{A}$). The algorithm is the same for all robots, and the result of the `Compute` phase is a destination point. Actually, for robots on graphs, the result of this phase either is the vertex where the robot currently resides or it is a vertex among those at one hop distance (i.e., at most one edge can be traversed);
4. `Move`. If the destination point is the current vertex where r resides, r performs a *nil* movement (i.e., it does not move); otherwise, it moves to the adjacent vertex selected.

When a robot is in `Wait`, we say it is *inactive*, otherwise it is *active*. In the literature, the computational cycle is simply referred to as the `Look-Compute-Move` (LCM) cycle, as during the `Wait` phase a robot is inactive.

Since robots are oblivious, they have no memory of past events. This implies that the `Compute` phase is based only on what is determined in their current cycle (in particular, from the snapshot acquired in the current `Look` phase). A data structure containing all the information elaborated from the current snapshot represents what later is called the *view* of

[1] No randomization features are allowed.

a robot. Since each robot refers to its own LCS, the view cannot exploit absolute orienteering but it is based on relative positions of robots.

Concerning the movements, in the graph environment moves are always considered as instantaneous. This results in always perceiving robots on vertices and never on edges during Look phases. Hence, robots cannot be seen while moving, but only at the moment they may start moving or when they arrived. Two or more robots can move toward the same vertex at the same time, thus creating what it called a *multiplicity* (i.e., a vertex occupied by more than one robot). When undesired, a multiplicity is usually referred to as a *collision*.

In the literature, different characterizations of the environment have been considered according to whether robots are fully-synchronous, semi-synchronous, or asynchronous (cf. [28,29]). These synchronization models are defined as follows:

- *Fully-Synchronous* (FSync): All robots are always active, continuously executing in a synchronized way their LCM-cycles. Hence the time can be logically divided into global rounds. In each round, all the robots obtain a snapshot of the environment, compute on the basis of the obtained snapshot and perform their computed move;
- *Semi-Synchronous* (SSync): robots are synchronized as in FSync but not all robots are necessarily activated during a LCM-cycle;
- *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 and Async cases, it is usually assumed the existence of an *adversary* which determines the computational cycle's timing. Such timing is assumed to be *fair*, that is, each robot performs its LCM-cycle within finite time and infinitely often. Without such an assumption the adversary may prevent some robots from ever moving.

It is worth remarking that the three synchronization schedulers induce the following hierarchy (see [30]): FSync robots are more powerful (i.e., they can solve more tasks) than SSync robots, that in turn are more powerful than Async robots. This simply follows by observing that the adversary can control more parameters in Async than in SSync, and more in SSync than in FSync. In other words, protocols designed for Async robots also work for SSync and FSync robots. On the contrary, any impossibility result proved for FSync robots also holds for SSync, and Async robots.

Whatever the assumed scheduler is, the activations of the robots according to any algorithm $\mathcal{A}$ determine a sequence of specific time instants $t_0 < t_1 < t_2 < \cdots$ during which at least one robot is activated. Apart from the Async case where the notion of time is not shared by robots, for the other types of schedulers robots are synchronized. In the FSync case, each robot is active at each time unit. In the SSync, we assume that at least one robot is active at each time t. If $C(t)$ denotes the configuration observed by some robots at time t during their Look phase, then an *execution* of $\mathcal{A}$ from an initial configuration C is a sequence of configurations $\mathbb{E} : C(t_0), C(t_1), \ldots$, where $C(t_0) = C$ and $C(t_{i+1})$ is obtained from $C(t_i)$ by moving at least one robot (which is active at time t_i) according to the result of the Compute phase as implemented by $\mathcal{A}$. Note that, in SSync or Async there exists more than one execution of $\mathcal{A}$ from $C(t_0)$ depending on the activation of the robots or the duration of the phases, whereas in FSync the execution is unique as it always involves all robots in all time instants.

3. Problem formulation and resolution methodology

The topology where robots are placed is represented by a simple and connected graph $G = (V, E)$. A function $\lambda : V \to \mathbb{N}$ gives the number of robots on each vertex of G, and we call $C = (G, \lambda)$ a *configuration* whenever $\sum_{v \in V} \lambda(v)$ is bounded and greater than zero. In this paper, we introduce the Geodesic Mutual Visibility (GMV, for short) problem:

Problem (GMV)**.**

Input: A configuration $C = (G, \lambda)$ in which each robot lies on a different vertex of a graph G.
Goal: Design a deterministic distributed algorithm working under the LCM model that, starting from C, brings all robots on distinct vertices – without generating collisions – in order to obtain the geodesic mutual visibility, that is there is a geodesic between any pair of robots where no other robots reside.

Since the definition of mutual visibility requires that robots are located in distinct vertices, then the above definition requires that any possible solving algorithm does not create multiplicities. In fact, as the robots are anonymous and homogeneous, regardless of the synchronicity model, the adversary will be able to keep the multiplicity unchanged and no algorithm will ever be able to separate the robots. It is worth remarking that this is a special case with respect to the general situation in which a configuration contains *equivalent* robots. The next paragraph provides a formal definition of such an equivalence relationship.

3.1. Symmetric configurations

Two undirected graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there is a bijection φ from V to V' such that $\{u, v\} \in E$ iff $\{\varphi(u), \varphi(v)\} \in E'$. 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 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 the graph G is called *asymmetric*, otherwise it is called *symmetric*. Two 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 isomorphism can be extended to configurations in a natural way: two configurations $C = (G, \lambda)$ and $C' = (G', \lambda')$ are isomorphic if G and G' are isomorphic via a bijection φ and $\lambda(v) = \lambda'(\varphi(v))$ for each vertex v in G. An *automorphism* on C is an isomorphism from C to itself and the set of all automorphisms of C forms a group that we call *automorphism group* of C and denote by $\mathrm{Aut}(C)$. Analogously to graphs, if $|\mathrm{Aut}(C)| = 1$, we say that the configuration C is *asymmetric*, otherwise it is *symmetric*. In a configuration C, two robots r_1 and r_2, respectively located on distinct vertices v_{r_1} and v_{r_2}, are *equivalent* if there exists $\varphi \in \mathrm{Aut}(C)$ such that $v_{r_2} = \varphi(v_{r_1})$. Note that $\lambda(v_{r_1}) = \lambda(v_{r_2})$ whenever v_{r_1} and v_{r_2} are equivalent.

From an algorithmic point of view, it is important to remark that when $\varphi \in \mathrm{Aut}(C)$ makes the elements of $V' \subseteq V$ pairwise equivalent, then a robot r_1 cannot distinguish its position $v_{r_1} \in V'$ from that of a robot r_2 located at vertex $v_{r_2} = \varphi(v_{r_1}) \in V'$. As a consequence, no algorithm can distinguish between two equivalent robots, and then it cannot avoid the adversary activates two equivalent robots at the same time and that they perform the same move simultaneously.

3.2. Methodology

The algorithms proposed in this paper are designed according to the methodology proposed in [3]. Assume that an algorithm $\mathcal{A}$ must be designed to resolve a generic problem $\mathcal{P}$. Here we briefly summarize how $\mathcal{A}$ can be designed according to that methodology.

In general, a single robot has rather weak capabilities with respect to $\mathcal{P}$ it is asked to solve along with other robots (we recall that robots have no direct means of communication). For this reason, $\mathcal{A}$ should be based on a preliminary decomposition approach: $\mathcal{P}$ should be divided into a set of sub-problems so that each sub-problem is simple enough to be thought as a "task" to be performed by (a subset of) robots. This subdivision could require several steps before obtaining the definition of such simple tasks, thus generating a sort of hierarchical structure. Assume now that $\mathcal{P}$ is decomposed into simple tasks $T_1, T_2, \ldots, T_k$, where one of them is the *terminal* one, i.e. the robots recognize that the current configuration is the one in which $\mathcal{P}$ is solved and do not make any moves.

According to the `LCM` model, during the `Compute` phase, each robot must be able to recognize the task to be performed just according to the configuration perceived during the `Look` phase. This recognition can be performed by providing $\mathcal{A}$ with a *predicate* P_i for each task T_i. Given the perceived configuration, the predicate P_i that results to be true reveals to robots that the corresponding task T_i is the task to be performed. With predicates P_i well-formed, algorithm $\mathcal{A}$ could be used in the `Compute` phase as follows: – *if a robot r executing algorithm $\mathcal{A}$ detects that predicate P_i holds, then r simply performs a move m_i associated with task T_i*. In order to make this approach valid, the well-formed predicates must guarantee the following properties:

Prop_1**:** each P_i must be computable on the configuration C perceived in each `Look` phase;

Prop_2**:** $P_i \wedge P_j = \mathtt{false}$, for each $i \neq j$; this property allows robots to exactly recognize the task to be performed;

Prop_3**:** for each possible perceived configuration C, there must exist a predicate P_i evaluated true.

Concerning the definition of the predicates, it is reasonable to assume that each task T_i requires some *precondition* to be verified. Hence, in general, to define the predicates we need:

- *basic variables* that capture metric/topological/numerical/ordinal aspects of the input configuration which are relevant for the used strategy and that can be evaluated by each robot on the basis of its view;
- *composed variables* that express the preconditions of each task T_i.

If we assume that $\mathtt{pre}_i$ is the composed variable that represents the preconditions of P_i, for each $1 \leq i \leq k$, then predicate P_i can be defined as follow:

$$P_i = \mathtt{pre}_i \wedge \neg(\mathtt{pre}_{i+1} \vee \mathtt{pre}_{i+2} \vee \ldots \vee \mathtt{pre}_k) \tag{1}$$

This definition ensures that any predicate fulfills Property Prop_2 (it is directly implied by Eq. (1)).

Consider now an execution of $\mathcal{A}$, and assume that a task T_i is performed with respect to the current configuration C. If $\mathcal{A}$ transforms C into C' and this new configuration has to be assigned the task T_j, then we say that $\mathcal{A}$ can generate a transition from T_i to T_j. The set of all possible transitions of $\mathcal{A}$ determines a directed graph called *transition graph*. Of course, the terminal task among $T_1, T_2, \ldots, T_k$ must be a sink node in the transition graph.

According to the proposed methodology, in [3] it is shown that the correctness of $\mathcal{A}$ can be obtained by proving that all the following properties hold:

H_1: for each task T_i, the tasks reachable from T_i by means of transitions are exactly those represented in the transition graph (i.e., the transition graph is correct);

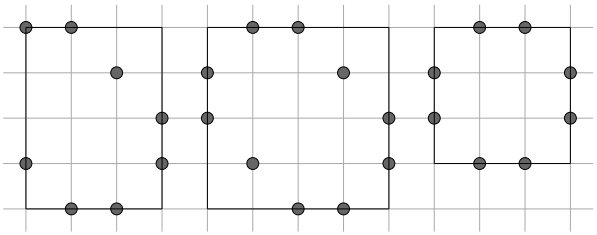


Fig. 1. Examples of $mbr(R)$.

H_2: possible cycles in the transition graph (including self-loops) must be performed a finite number of times – apart for the self-loop induced by a terminal task;

H_3: unsolvable configurations are not generated by $\mathcal{A}$ (with respect to GMV, for instance, this means that $\mathcal{A}$ does not generate multiplicities, i.e., it is collision-free).

4. Solving GMV on square grids

In this section, we solve GMV for robots moving on finite or infinite square grids embedded in the plane. Moreover, we add the further requirement to obtain a placement of the robots so as that the final minimum bounding rectangle enclosing all the robots is of minimum area. This area-constrained GMV problem is denoted as GMV_{area}. Such a requirement avoids 'trivial' solutions – in terms of feasibility, especially in the case of infinite grids. In fact, by aligning all the robots along a diagonal, GMV would be solved. In a finite grid G, instead, this is not always possible, depending on the number of robots. Hence, different approaches are required anyway. Moreover, when the number of robots is exactly $\mu(G)$, then the minimum area constraint is forced.

While considering synchronous robots endowed with chirality (i.e., they share a common handedness), we provide time-optimal algorithms solving GMV_{area} in both finite and infinite grid graphs.

4.1. Preliminary concepts and notation

Given a graph G, let $d(u, v)$ be the distance in G between two vertices $u, v \in V$ in terms of the minimum number of edges traversed. We extend the notion of distance to robots: given $r_i, r_j \in R$, $d(r_i, r_j)$ represents the distance between the vertices in which the robots reside. $D(r)$ denotes the sum of distances of $r \in C$ from any other robot, that is $D(r) = \sum_{r_i \in C} d(r, r_i)$. A square tessellation of the Euclidean plane is the covering of the plane using squares of side length 1, called tiles, with no overlaps and in which the corners of squares are identically arranged. Let S be the infinite lattice formed by the vertices of the square tessellation. The graph called infinite **grid graph**, and denoted by G_∞, is such that its vertices are the points in S and its edges connect vertices that are distance 1 apart. In this section, G denotes a finite grid graph formed by $M \cdot N$ vertices (i.e., informally generated by M "rows" and N "columns"). By $mbr(R)$, we denote the **minimum bounding rectangle** of R, that is the smallest rectangle (with sides parallel to the edges of G) enclosing all the robots (cf. Fig. 1). Note that $mbr(R)$ is unique. By $c(R)$, we denote the center of $mbr(R)$.

Symmetric configurations. As chirality is assumed, then the only possible symmetries that can occur in our setting are rotations of 90 or 180 degrees. A rotation is defined by a center c and a minimum angle of rotation $\alpha \in \{90, 180, 360\}$ working as follows: if the configuration is rotated around c by an angle α, then a configuration coincident with itself is obtained. The **order** of a configuration is given by $360/\alpha$. A configuration is **rotational** if its order is 2 or 4. The **symmetricity** of a configuration C, denoted as $\rho(C)$, is equal to its order, unless its center is occupied by one robot, in which case $\rho(C) = 1$. Clearly, any asymmetric configuration C implies $\rho(C) = 1$.

The **type of center** of a rotational configuration C is denoted by $tc(C)$ and is equal to 1, 2, or 3 according to whether the center of rotation is on a vertex, on a median point of an edge, or on the center of a square of the tessellation forming a grid, respectively (cf. Fig. 2).

The view of robots. In $\mathcal{A}$, robots encode the perceived configuration into a binary string called **lexicographically smallest string** and denoted as $LSS(R)$ (cf. [5,6]). To define how robots compute the string, we first analyze the case in which $mbr(R)$ is a square: the grid enclosed by $mbr(R)$ is analyzed row by row or column by column starting from a corner and proceeding clockwise, and 1 or 0 corresponds to the presence or the absence, respectively, of a robot for each encountered vertex. This produces a string assigned to the starting corner, and four strings in total are generated. If $mbr(R)$ is a rectangle, then the approach is restricted to the two strings generated along the smallest sides. The lexicographically smallest string is the $LSS(R)$. Note that, if two strings obtained from opposite corners along opposite directions are equal, then the configuration is rotational, otherwise it is asymmetric. The robot(s) with **minimum view** is the one with minimum position in $LSS(R)$. The first three configurations shown in Fig. 1 can be also used for providing examples about the view.

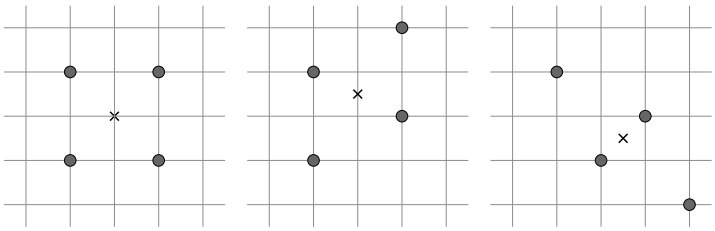


Fig. 2. Examples for the notion of center of three rotational configurations: in order, $tc(C_1) = 1$, $tc(C_2) = 2$, and $tc(C_3) = 3$.

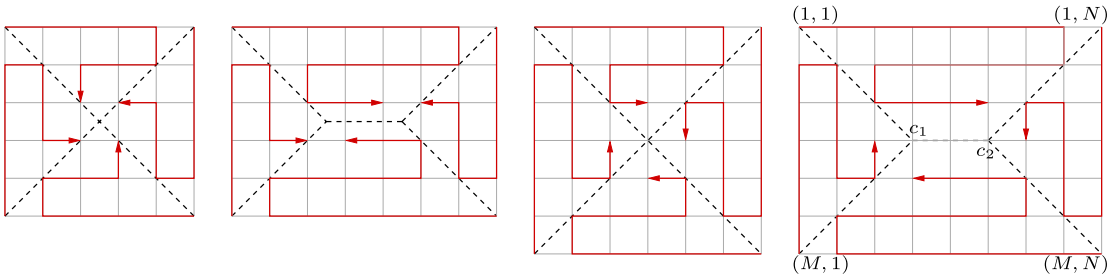


Fig. 3. Examples of special-paths with respect to different configurations.

In particular: in the first case, we have $\rho(C) = 1$ and $LSS(R) = 0110\ 1001\ 1000\ 0100\ 0011$, where the string starts from the bottom-right corner; in the second case, we have $\rho(C) = 2$ and $LSS(R) = 00110\ 01001\ 10001\ 10010\ 01100$, where the string starts from the bottom-left or the top-right corners; in the last case, we have $\rho(C) = 4$ and $LSS(R) = 0110\ 1001\ 1001\ 0110$, where the string starts from any corner.

Regions. Our algorithms assume that robots are assigned to **regions** of $mbr(R)$ as follows (cf. Fig. 3). If $mbr(R)$ is a square, the four regions are those obtained by drawing the two diagonals of $mbr(R)$ that meet at $c(R)$. If $mbr(R)$ is a rectangle, then from each of the vertices positioned on the shorter side of $mbr(R)$ starts a line at 45 degrees toward the interior of $mbr(R)$ – these two pairs of lines meet at two points (say $c_1(R)$ and $c_2(R)$) which are then joined by a segment.

In each of the four regions, it is possible to define a **special-path** that starts from a corner v and goes along most of the vertices in the region. To simplify the description of such a path, assume that $mbr(R)$ coincides with a sub-grid with M rows and N columns, and the vertices are denoted as (i, j), with $1 \leq i \leq M$ and $1 \leq j \leq N$. The special-path that starts at $(1, 1)$ is made of a sequence of "traits" defined as follows: the first trait is $(1, 1), (1, 2), \ldots, (1, N-1)$, the second is $(2, N-1), (2, N-2), \ldots, (2, 3)$, the third is $(3, 3), (3, 4), \ldots, (3, N-3)$, and so on. This process ends after $\lfloor \min\{M, N\}/2 \rfloor$ traits are formed in each region, and the special-path is obtained by composing, in order, the traits defined in each region (see the red lines in Fig. 3).

4.2. An algorithm for GMV_{area}

In this section, we present a resolution algorithm for the GMV_{area} problem, when considering $n \geq 7$ fully synchronous robots endowed with chirality and moving on a finite grid graph G with $M, N \geq \lceil \frac{n}{2} \rceil$ rows and columns. Note that the constraints on the number of rows and columns depend on the fact that on each row (or column) it is possible to place at most two robots, otherwise the outermost robots on the row (or column) are not in mutual visibility. Concerning the number of robots, we omit the cases with $n < 7$ as they require just tedious and specific arguments that cannot be generalized. Hence, we prefer to cut them out of the discussion, even though they can be solved.

Our approach is to first design a specific algorithm $\mathcal{A}_{\text{asym}}$ that solves GMV_{area} only for **asymmetric configurations**. Later, we will describe (1) how $\mathcal{A}_{\text{asym}}$ can be extended to a general algorithm $\mathcal{A}$ that also handles symmetric configurations, and (2) how, in turn, $\mathcal{A}$ can be modified into an algorithm $\mathcal{A}_\infty$ that solves the same problem for each input configuration defined on infinite grids.

The pattern formation approach. $\mathcal{A}_{\text{asym}}$ follows the "pattern formation" approach. In the general pattern formation problem, robots belonging to an initial configuration C are required to arrange themselves in order to form a configuration F which is provided as input. In [31,32], it is shown that F can be formed if and only if $\rho(C)$ divides $\rho(F)$. Hence, here we show some patterns that can be provided as input to $\mathcal{A}_{\text{asym}}$ so that:

1. $\rho(C)$ divides $\rho(F)$;
2. if $\rho(C) \in \{2, 4\}$ then $tc(C) = tc(F)$;
3. the positions specified by F solve GMV_{area}.

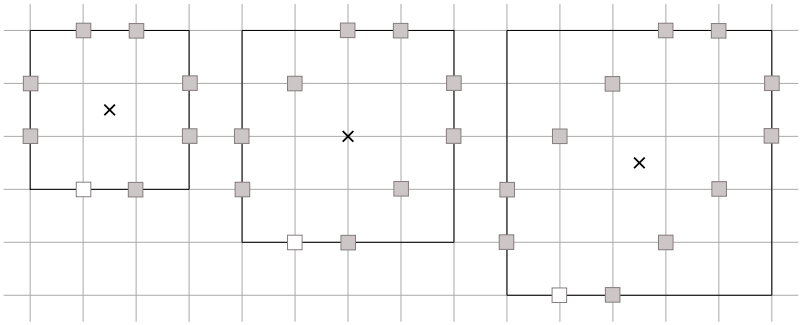


Fig. 4. Patterns F for asymmetric input configurations with $n = 8, 10, 12$ robots. For $n = 7, 9, 11$, the position represented in white is not considered in F.

The first requirement trivially holds since we are assuming that C is asymmetric and hence $\rho(C) = 1$. The second is required since the center of symmetric configurations is an invariant for synchronous robots. Concerning the last requirement, in Fig. 4 we show some examples for F when $7 \leq n \leq 12$. In [25], it is shown how F is defined for any n and it is also proved that the elements in these patterns always solve GMV for the grid G. Finally, since in F there are two robots per row and per column, and since in $mbr(F)$ all the rows and columns are occupied (for n even), it can be easily observed that F solves GMV_{area}.

High level description of the algorithm. The algorithm is designed according to the methodology recalled in Section 3.2 that allows dividing the problem GMV_{area} into a set of sub-problems that are simple enough to be thought as "tasks" to be performed by (a subset of) robots.

As a first sub-problem, the algorithm $\mathcal{A}_{\text{asym}}$ selects a single robot, called guard r_g, to occupy a corner of the grid G. As robots are disoriented (only sharing chirality), the positioning of the guard allows the creation of a common reference system used by robots in the successive stages of the algorithm. Given chirality, the position of r_g allows robots to identify and enumerate rows and columns. r_g is not moved until the final stage of the algorithm and guarantees that the configuration C is kept asymmetric during the movements of the other robots. Given the common reference system, all robots agree on the embedding of the pattern F, which is realized by placing the corner of F with the maximum view in correspondence with the corner of G in which r_g resides. This sub-problem is solved by tasks T_{1a}, T_{1b}, or T_{1c}. In task T_2, the algorithm moves the robots so as to obtain the suitable number of robots for each row according to pattern F, that is, two robots per row. The only exception comes when n is odd, in which case the last row will require just one robot. During task T_3, robots move toward their final target along rows, except for r_g. When T_3 ends, $n - 1$ robots are in place according to the final pattern F. During task T_4, r_g moves from the corner of G toward its final target, placed on a neighboring vertex, hence leading to the final configuration in one step.

4.3. Detailed description of the tasks

In this section, we provide all the necessary details for each of the designed tasks.

Task T_1. Here the goal is to select a single robot r_g to occupy a corner of the grid G. This task is divided into three sub-tasks based on the number of robots occupying the perimeter – and in particular the corners, of G. Let RS be the number of robots on the sides of G, and let RC be the number of robots on the corners of G.

Task T_{1a} starts when there are no robots on the perimeter of G and selects the robot r_g such that $D(r)$ is maximum, with r of minimum view in case of ties. The planned move is m_{1a}: *r_g moves toward the closest side of G.* At the end of the task, r_g is on the perimeter of G.

Task T_{1b} activates when the following precondition holds:

$$\texttt{pre}_{1b} \equiv RS \geq 1 \wedge RC = 0.$$

In this case, there are at least two robots on the perimeter of G but none on corners. The task selects the robot r_g located on a side of G closest to a corner of G, with the minimum view in case of ties, to move toward a corner of G. Move m_{1_b} is defined as follows: *r_g moves toward the closest corner of G* – arbitrarily chosen if more than one. At the end of task T_{1b}, a single robot r_g occupies a corner of the grid G.

Task T_{1c} activates when the following precondition holds:

$$\texttt{pre}_{1c} \equiv RC > 1.$$

In this case, all the robots on the corners but one move away from the corners. The moves are specified by Algorithm 1. This algorithm uses some additional definitions. In particular, a special-path is said **occupied** if there is a robot on its

Algorithm 1 `MoveAlong` special-path

Input: a configuration C
1: **if** $p = 0$ **then**
2: Let S be the occupied special-path whose first robot has the minimum view.
3: **move**: all the robots on a special-subpath and not on S move toward the neighbor vertex along the special-path.
4: **if** $p = 1$ **then**
5: Let I be the fully-occupied special-path
6: **move**: all the robots on a special-subpath and not on I move toward the neighbor vertex along the special-path
7: **if** $p = 2$ **then**
8: **move**: the robot on a corner of G, with an empty neighbor, moves toward it.

corner. A special-path is said to be **fully-occupied** if robots are placed on all its vertices. Given an occupied special-path P, a special-subpath is a fully occupied sub-path of P starting from the corner of P. Finally, p denotes the number of fully-occupied special-paths.

At line 1, the algorithm checks if there are no fully-occupied special-paths. In this case, there are at least two occupied special-paths. The robot, occupying the corner, with minimum view, is elected as guard r_g. The move is designed to empty all the other corners of G except for the one occupied by r_g. In each occupied special-paths, but the one to which r_g belongs to, the robots on the corners, and those in front of them along the special-paths until the first empty vertex, move forward along the special-path. At line 4, there is exactly one fully-occupied special path. Therefore, robots on the fully-occupied special-path are kept still. Concerning the other occupied special-paths, the robots on corners, and those in front of them until the first empty vertex, move forward along the special-path. At line 7 there is more than one fully-occupied special-path. Actually, this condition can occur only for a 4×4 grid G with two fully-occupied special-paths located on two successive corners of G. Therefore, there is a single robot r, on a corner of G, with an empty neighbor. Then, r moves toward that neighbor.

Note that, Algorithm 1 is designed so that, in a robot cycle, a configuration is obtained where exactly one corner of G is occupied.

Task T_2. In task T_2, the algorithm moves the robots to place the suitable number of robots for each row according to the pattern F, starting from the first row, while possible spare rows remain empty. At the end of the task, for each row corresponding to those of the pattern F, there are two robots, except when the number of robots n is odd, in which case in the last row is placed a single robot. The position of r_g allows robots to identify the embedding of F and hence the corresponding rows and columns. We assume, without loss of generality, that r_g is positioned on the upper-right corner of G. r_g identifies the first row. In this task, we define $c(r)$ and $l(r)$ as the column and the row, respectively, where robot r resides. Columns are numbered from left to right, therefore $l(r_g) = 1$ and $c(r_g) = N$. Let t_l be the number of targets on row l in F, let $(t_1, t_2, \ldots, t_M)$ be the vector of the number of targets, and let $(n_1, n_2, \ldots, n_M)$ be the number of robots on each of the M rows of G.

For each row l, the algorithm computes the number of exceeding robots above and below l with respect to the number of targets, to determine the number of robots that need to leave row l. Given a row l, let R_l be the number of robots on rows from 1 to $l-1$, and let R'_l be the number of robots on rows from $l+1$ to M. Accordingly, let $\hat{t}_l$ and $\check{t}_l$ be the number of targets above and below the line l, respectively. We define the subtraction operation $\dot{-}$ between two natural numbers a and b as $a \dot{-} b = 0$ if $a < b$, $a \dot{-} b = a - b$, otherwise. Concerning to the number of targets, given a row l, let B_l be the number of exceeding robots above l, l included, and let A_l be the number of exceeding robots below l, l included. Formally, $B_l = (R_l + n_l) \dot{-} (\hat{t}_l + t_l)$ and $A_l = (R'_l + n_l) \dot{-} (\check{t} + t_l)$.

Let $RD_l = n_l - (n_l \dot{-} B_l)$ be the number of robots that must move downward and $RU_l = n_l - (n_l \dot{-} A_l)$ be the number of robots that must move upward from row l. Task T_2 activates when precondition $\texttt{pre}_2$ becomes true:

$$\texttt{pre}_2 \equiv RC = 1 \wedge \exists\ l \in 1, \ldots, M : B_l \neq 0 \vee A_l \neq 0.$$

The precondition identifies the configuration in which the guard r_g is placed on a corner of G and there is at least a row in which there is an excess of robots. We define **outermost** any robot that resides on the first or the last column of G. Let U_l (D_l, resp.) be a set of robots on row l chosen to move upward (downward, resp.) and let U (D, resp.) be the list of sets U_l (D_l, resp.) with $l \in \{1, \ldots, M\}$. The robots that move upward or downward are chosen as described in Algorithm 2.

For each row l, at lines 4–7, the algorithm computes the number of exceeding robots B_l, A_l, and the number of robots that must leave the row RD_l and RU_l. Then, it checks whether the number M of rows of G is greater than the number k of rows of F. The algorithm selects RD_l robots to move downward, starting from the first column, and A_l robots to move upward, starting from the Nth column.

Line 11 corresponds to the case in which $M = k$, the algorithm selects RD_l robots to move downward, starting from the second column and RU_l robots to move upward, starting from the $N-1$ column. This avoids the selection of robots that may move in one of the corners of G. At line 14, the algorithm checks if a robot r selected to move upward on row 2, occupies vertex (2, 1). In the positive case, r is removed from U_2. This avoids r to move to a corner of G. At line 15, the algorithm returns the sets U of robots chosen to move upward for each row, and the sets D of robots chosen to move

Algorithm 2 `SelectRobots`

Input: $C' = (C \setminus r_g)$
1: Let $U = \{U_1, U_2, \ldots, U_M\}$ be a list of empty sets
2: Let $D = \{D_1, D_2, \ldots, D_M\}$ be a list of empty sets
3: **for all** $l \in (1 \ldots m)$ **do**
4: $B_l \leftarrow (R_l + n_l) \dot{-} (\hat{t}_l + t_l)$
5: $A_l \leftarrow (R'_l + n_l) \dot{-} (\check{t}_l + t_l)$
6: $RD_l \leftarrow n_l - (n_l \dot{-} B_l)$
7: $RU_l \leftarrow n_l - (n_l \dot{-} A_l)$
8: **if** $M > \lceil n/2 \rceil$ **then**
9: Let U_l be the set of RU_l robots of row l selected from right
10: Let D_l be the set of RD_l robots of row l selected from left
11: **else**
12: Let U_l be the set of RU_l robots of row l from right and not outermost
13: Let D_l be the set of RD_l robots of row l from left and not outermost
14: **if** $U_2 = \{r\}$ and $l(r) = 2$ and $c(r) = 1$ **then** $U_2 = \emptyset$
15: **return** U, D

Algorithm 3 `MoveRobot`

Input: a configuration C, guard r_g
1: U, D = `SelectRobots`$(C \setminus r_g)$
2: **for all** robots r **do**
3: Compute $t(r)$
4: **if** $r \notin U_{l(r)}$ or $\forall\, r_1, r_2,\ t(r_1) \neq t(r_2)$ **then**
5: move to $t(r)$

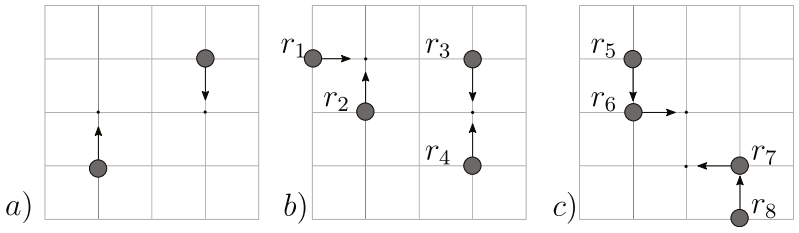


Fig. 5. The three possible movement combinations as described in task T_2. Gray circles represent robots, arrows represent the direction of movements, and small dots are robot targets.

downward. Given a robot r on a row l, let *AlignedUp* be the Boolean variable that is true when there exists another robot r' such that ($U_{l+1} = \{r'\}$ and $c(r) = c(r')$) holds, and *AlignedDown* be the Boolean variable that is true when there exists another robot r'' such that ($D_{l-1} = \{r''\}$ and $c(r) = c(r'')$) holds. Let $t(r)$ be the target of a robot r defined as follows:

$$t(r) = \begin{cases} (l(r)+1, c(r)) & \text{if } r \in D_l \\ (l(r)-1, c(r)) & \text{if } r \in U_l \\ (l(r), c(r)-1) & \text{if (}\mathit{AlignedUp} \text{ or } \mathit{AlignedDown}\text{) and } c(r) \geq N/2 \\ (l(r), c(r)+1) & \text{if (}\mathit{AlignedUp} \text{ or } \mathit{AlignedDown}\text{) and } c(r) < N/2 \\ (2,2) & \text{if } RU_2 = 1 \text{ and } \exists!\ \text{r on } l_2 \mid c(r) = 1 \text{ and } l(r) = 2 \\ (l(r), c(r)) & \mathit{otherwise} \end{cases} \tag{2}$$

The first two cases reported in the definition of Eq. (2) identify the target of robot r when is selected to move downward (upward, resp.). The target of r is one row below (above, resp.) its current position and on the same column. The third and the fourth cases refer to the occurrence in which there is a robot r_1, positioned in the same column of r, that is selected to move upward or downward. Then, the target of r is on a neighboring vertex, on the same row, closer to the center of G. The fifth case reports the target of a robot r when positioned on the second row and first column, and one robot is required to move on the first row. To avoid occupying a corner of G, the target of r is the neighboring vertex to r on its same row. In all other cases, the target of a robot r is its current position. Robots move according to Algorithm 3.

Each robot runs Algorithm 3 independently. At line 1, a robot calls procedure `SelectRobots`on $C' = \{C \setminus r_g\}$ and acquires the sets of robots selected to move upward and downward, respectively. At lines 2–3, a robot computes the targets of all the robots. At line 4, the robot checks if it is not selected to move upward and if any couple of robots have the same target. This test avoids collisions. Possible conflicting moves are shown in Fig. 5(b). Two robots can have the same target when they are in the same column at distance two and the robot with the smallest row index is selected to

Table 1
The table summarizes the phases of the algorithm: the first column reports a summary of the task's goal, the second column reports the task's name, the third column reports, for each task the precondition to enter the task, the last column reports the transitions among tasks.

Sub-problem	Task	Precondition	Transitions
Placement of the guard robot	T_{1a}	True	T_{1a}, T_{1b}
	T_{1b}	$RS \geq 1 \wedge RC = 0$	T_{1b}, T_2, T_3, T_4
	T_{1c}	$RC > 1$	T_2, T_3, T_4
Bringing t_l robots for each row	T_2	$RC = 1 \wedge \exists\, l \in \{1 \dots m\}$: $B_l \neq 0 \vee A_l \neq 0$	T_2, T_3, T_4
Bring $n-1$ robots to final target	T_3	$RC = 1 \wedge \forall$ row l $(B_l = 0 \wedge A_l = 0)$	T_3, T_4
Bring the guard robot to final target	T_4	$n-1$ robots on final target	T_5
Termination	T_5	F formed	T_5

move downward, while the other upward. An example is shown in Fig. 5(b) for robots r_3 and r_4. The only other possible collision is for the robot r_1 having $t(r_1) = (2, 2)$ (case five in Eq. (2)). There might be a robot r_2 with $l(r_2) = 3$ and $c(r_2) = 2$ selected to move upward. This configuration is shown in Fig. 5(b). In all these cases, to avoid any collision, the upward movement is performed only when there are no robots having the same target, otherwise the robot stays still. Each conflict is resolved in a robot cycle since downward and side movements are always allowed.

Fig. 5 shows the three types of possible movements performed by robots. Robots move concurrently without collisions. Fig. 5(a) shows robots moving downward or upward and having different targets. Fig. 5(b) shows two robots having the same target. To resolve the conflict, the upward movement is stopped for a cycle. Fig. 5(c) shows the cases in which a robot is selected to move upward (r_8) or downward (r_5) on a target vertex that is already occupied by another robot (r_7, r_6 respectively). Robots r_5 and r_8 perform their move while r_6 and r_7 move on a neighboring vertex on the same row and closer to the center of G. Since movements are concurrent (robots are synchronous), collisions are avoided.

Task T_3. This task is designed to bring $n-1$ robots to their final target except for r_g. This task activates when task T_2 is over, therefore $\texttt{pre}_3$ holds:

$$\texttt{pre}_3 \equiv RC = 1 \wedge \forall \text{ row } l : (B_l = 0 \wedge A_l = 0)$$

Given the embedding of F on G, in each row l, there are t_l targets and n_l robots, with $t_l = n_l$, therefore robots identify their final target and move toward it without collisions. Given the particular shape of F, there are at most two targets per row, therefore we can state the move m_3 as follows: *for each row, the rightmost robot moves toward the rightmost target and the leftmost robot moves toward the leftmost target except for r_g.*

Task T_4. During task T_4, the guard r_g moves from the corner of G and goes toward its final target. This task activates when $\texttt{pre}_4$ holds:

$$\texttt{pre}_4 \equiv n-1 \text{ robots but } r_g \text{ match their final target.}$$

The corresponding move is called m_4 and is defined as follows: *r_g moves toward its final target*. The embedding of F guarantees that the final target of r_g is on its neighboring vertex on row 1. Therefore, in one step, r_g reaches its target. After task T_4, the pattern is completed.

Task T_5. This is the task in which each robot recognizes that the pattern is formed and no more movements are required. Each robot performs the nil movement keeping the current position. The precondition is

$$\texttt{pre}_5 \equiv F \text{ is formed.}$$

Although our algorithm is designed so as to form a specific pattern F that solves GMV_{area}, $\texttt{pre}_5$ could be simply stated as 'GMV_{area} solved'. In this way, robots would stop moving as soon as the problem is solved and not necessarily when the provided pattern F is formed. However, since the formation of F is usually required, for the ease of the discussion we prefer the current form for $\texttt{pre}_5$.

4.4. Formalization and correctness

We have already remarked that the algorithm has been designed according to the methodology recalled in Section 3.2. Accordingly, Table 1 summarizes the designed tasks, the corresponding preconditions, and the possible transitions from each task. Furthermore, all the transitions are shown in the transition graph depicted in Fig. 6.

We observe that the predicates used in the algorithm are all well-formed since they guarantee that Prop_1, Prop_2, and Prop_3 are all valid. In particular, Prop_1 follows from the definition of the simple preconditions expressed in Table 1, Prop_2 holds because each predicate P_i has been defined as indicated in Eq. (1), and Prop_3 directly follows from the definitions of P_i (if $P_5, P_4, \dots, P_{1b}$ are all false, then P_{1a} holds).

Concerning the correctness of $\mathcal{A}_{\text{asym}}$, still using the methodology in the remainder of this section we show that properties H_1, H_2, and H_3 hold by providing a specific lemma for each task. Finally, such lemmata will be used in a final theorem responsible for assessing the correctness of $\mathcal{A}_{\text{asym}}$.

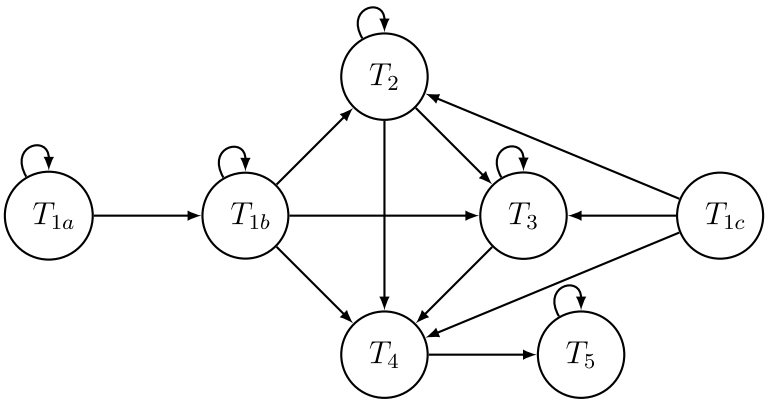


Fig. 6. Transition graph (derived from Table 1).

Lemma 1. *Let C be a configuration in T_{1a}. From C, $\mathcal{A}_{\mathsf{asym}}$ eventually leads to a configuration belonging to T_{1b}.*

Proof. In task T_{1a}, Algorithm $\mathcal{A}_{\mathsf{asym}}$ selects a robot denoted as r_g, called guard, such that $D(r)$ is maximum and with the minimum view in case of ties. Let us analyze properties H_i, for $1 \leq i \leq 3$, separately.

H_1**:** In task T_{1a}, no robots are on a side of the grid G, nor on its corners and r_g moves toward the closest side of G, $D(R)$ increases for r_g, therefore, r_g is repeatedly selected. When r_g reaches a side of G, it is the only robot on a side of G and $RS = 1$. Still, there are no robots on corners of G therefore $RC = 0$, $\texttt{pre}_{1b}$ becomes `true` and the configuration is in T_{1b}, since the preconditions of all the other tasks, except for T_5, require at least one robot on a corner of G, and T_5 requires more than one robot on the sides of G.

H_2**:** At each cycle, r_g decreases its distance from the closest side of $mbr(C)$ by one. Therefore, within a finite number of `LCM` cycles, it reaches its target and the configuration is not in T_{1a} anymore.

H_3**:** Since r_g is the robot such that $D(r)$ is maximum, it must be on a side of $mbr(C)$. While moving toward the closest side of G, r_g increases its distance from the other robots therefore it cannot meet any other robot on its way toward the target and no collision can occur. □

Lemma 2. *Let C be a configuration in T_{1b}. From C, $\mathcal{A}_{\mathsf{asym}}$ eventually leads to a configuration belonging to T_2, T_3 or T_4.*

Proof. In task T_{1b}, Algorithm $\mathcal{A}_{\mathsf{asym}}$ selects a robot denoted as r_g on the perimeter of G, closest to a corner of G, and having the minimum view in case of ties. Let us analyze properties H_i, for $1 \leq i \leq 3$, separately.

H_1**:** At the beginning of the task, there are no robots on a corner of G, and r_g moves toward the closest corner. As r_g moves toward its target, the distance from it decreases, therefore r_g is repeatedly selected. When it reaches its target, there is a single robot on a corner of G and $RC = 1$. Then, the obtained configuration can be in T_2, T_3 or T_4, all configurations in which the r_g is placed on a corner of G. The obtained configuration is not in T_5 because the pattern F has no targets on the corners of G.

H_2**:** At each cycle, r_g decreases its distance from the closest corner of G by one. Therefore, within a finite number of `LCM` cycles, r_g reaches its target and the configuration is not in T_{1b} anymore.

H_3**:** Since r_g is the robot closest to the corner of G it cannot meet any other robot on its way toward the target and no collision can occur. □

Lemma 3. *Let C be a configuration in T_{1c}. From C, $\mathcal{A}_{\mathsf{asym}}$ eventually leads to a configuration belonging to T_2, T_3 or T_4.*

Proof. In task T_{1c}, Algorithm 1 moves robots along special-paths. Let p be the number of fully-occupied special-paths. p cannot be greater than two and it can be two only when $k = \min(N, M) = 4$. In fact, the length of a special-path is $k^2/4$ when k is even and $(k^2 - 1)/4$ when k is odd, whereas the maximum number of robots is $2k$. For k even, we have that $pk^2/4 = 2k$, that is $pk = 8$. Hence, if $k > 4$ there can be only one fully-occupied special-path, otherwise $k = 4$ and there can be two fully-occupied special-paths. Similar analysis can be done for k odd that leads to $pk < 8$, then there can be only one fully-occupied special-path.

When $p = 2$, the special-paths must be on successive corners of G otherwise the configuration would be symmetric. Let us analyze properties H_i, for $1 \leq i \leq 3$, separately.

H_1: When T_{1c} starts, $RC \geq 1$. After the move, the guard r_g is placed and $RC = 1$. Therefore, the configuration is either in T_2, T_3 or T_4 and it is not in T_5 because the pattern F has no targets on corners of G.

H_2: In task T_{1c}, all the corners of G but one are emptied in a robot cycle.

H_3: The special-paths are designed so that they are disjoint. During task T_{1c}, only robots on special-subpaths move along the special-path. These are the robots on a corner of G and the ones in front of it until the first empty vertex. Since robots are synchronous, all these robots move forward by an edge, hence no collision can occur. □

Lemma 4. *Let C be a configuration in T_2. From C, $\mathcal{A}_{\mathsf{asym}}$ eventually leads to a configuration belonging to T_3 or T_4.*

Proof. During task T_2, robots move to place two robots per row. The only exception occurs when n is odd, in which case the last row requires just one robot. In particular, each robot runs Algorithm 3 in which they recall Algorithm 2 that selects the robots moving upward and downward for each row. The first row is identified by the position of r_g on the upper-right corner of G. Let us analyze properties H_i, for $1 \leq i \leq 3$, separately.

H_1: The choice of robots and their movements avoid robots occupying more than a corner of G. Indeed, Algorithm 2 selects robots moving upward and downward. When the number of rows M of the grid G are equal to $\lceil n/2 \rceil$, the algorithm selects robots between the second and the $(N-1)$-th column. The number of robots on the grid ensures that, even in a configuration in which robots in each row, from the second to the $(M-1)$-th one, occupy the first and the last columns, there are at least other two robots if n is odd and three if n is even that can be selected to move, able to finalize task T_2. Since no robots can move on a corner of G, then the configuration is not in T_{1a}, T_{1b} nor in T_{1c}.

When $M > \lceil n/2 \rceil$, robots do not move toward the last row of G, therefore they cannot occupy the corners of the Mth row of G.

If a robot r_1 occupies the vertex with coordinates $(2, 1)$, $RU_2 = 1$, and it is the only robot on row 2, to avoid occupying the corner of G with coordinates $(1, 1)$, the target of r_1 is $(2, 2)$ according to the fifth case of Eq. (2). If a robot r_2 occupies the vertex with coordinates $(2, N)$ and it is selected to move upward, r_2 moves on its target $(1, N)$ while the guard robot r_g moves to coordinates $(1, N-1)$ according to the third case of Eq. (2). Then, the role of r_g is taken by robot r_2 and a single corner of G is occupied by a robot. In both cases, the configuration is not in T_{1_c} because a single corner of G remains occupied. Moreover, the configuration is neither in T_{1a} nor T_{1b} since r_g is not moved, except for the case in which it is replaced by another robot.

During task T_2, the guard r_g is placed on a corner of G and $RC = 1$. At each cycle, B_l becomes 0 for the first row l for which $B_l \neq 0$. In at most $M-1$ steps, $B_l = 0$ and $A_l = 0$ for each l in at most $2(M-1)$ cycles, given that the upward movement can be prevented for a cycle when two robots have the same target. Examples of robots having the same target are depicted in Fig. 5(b). In both cases, the algorithm stops any upward movement, while allowing side and downward movements, see line 4 of Algorithm 3. At the successive cycle, robots are on the same column and both move. Once solved, no other conflict can occur in the same row. Then, in a finite number of cycles, A_l becomes 0 for each l. At the end of the task, there are two robots on each row except when n is odd, in which case the last row contains a single robot. Precondition $\mathtt{pre}_3$ becomes `true`, eventually, and the configuration is in T_3. If $n-1$ robots match their target, the configuration is in T_4 and it is not in T_5 because the pattern F has no targets on corners of G.

H_2: As described in H_1, at the end of this task, $B_l = 0 \wedge A_l = 0$ for each row l. This condition is reached in at most $2(M-1)$ cycles since the upward and downward movements are concurrent and no other configuration will be in T_2 anymore.

H_3: When the number of robots selected to move downward on row l is such that $RD_l \geq 2$, the exceeding number of robots on l will saturate all the targets of row $l+1$. Therefore, in the same cycle, any robots on row $l+1$ occupying the targets of robots on row l must also move downward. As a consequence, any robot selected to move downward on row l will reach a free target. When $RD_l = 1$, a robot r moves on row $l+1$ and at the same time, the robots on row $l+1$ will also move downward leaving at most one robot r_1. If r is on the same column of robot r_1, r moves downward while r_1 moves to its neighbor closer to the center of G (see Fig. 5(c)). Note that, the neighbors of r_1 will be empty since all other robots on row $l+1$ left the row. Moreover, the choice of the neighbor toward the center avoids r_1 going to one of the corners of G, see cases three and four of Eq. (2). The same reasoning applies to robots moving upward. When there are robots having the same target, see robots in Fig. 5(b) for reference, the algorithm detects this condition at line 4, and the upward movement is not performed. The robots are allowed to move downward or to the side, therefore collisions are avoided. □

Lemma 5. *Let C be a configuration in T_3. From C, $\mathcal{A}_{\mathsf{asym}}$ eventually leads to a configuration belonging to T_4.*

Proof. Task T_3 is designed to bring $n-1$ robots to their final target on F except for r_g. Let us analyze properties H_i, for $1 \leq i \leq 3$, separately.

H_1: During this task, there are r_l robots and t_l targets per row. In each row, robots move toward their final target on their same row. The task continues until $n-1$ robots are correctly placed according to the pattern, $\texttt{pre}_4$ becomes true and the configuration is in T_4.

H_2: At each LCM cycle, in each row, robots reduce the distance from their target by one until they reach the target.

H_3: There are at most two robots per row and two targets per row. Therefore, the rightmost robot goes to the rightmost target and the leftmost robot goes toward the leftmost target. In this way, collisions are avoided. □

Lemma 6. *Let C be a configuration in T_4. From C, $\mathcal{A}_{\text{asym}}$ eventually leads to a configuration belonging to T_5.*

Proof. In task T_4, $n-1$ robots are correctly positioned according to the pattern except for r_g. From this configuration, r_g moves toward its final target, in a single LCM cycle. Let us analyze properties H_i, for $1 \leq i \leq 3$, separately.

H_1: As r_g moves, it matches its target on F, then the pattern is formed, $\texttt{pre}_5$ becomes true and the configuration is in T_5.

H_2: The embedding of the pattern F guarantees that the target of r_g is at distance one from the corner of G in which it resides, therefore in one LCM cycle the task is over.

H_3: All robots, except for r_g, are matched and perform the nil movement, no other robots are on the target of r_g given the definition of m_4, therefore no collision can occur. □

In the following, we state our main result in terms of time required by the algorithm to solve the problem GMV_{area}. Time is calculated using the number of required LCM cycles given that robots are synchronous. Let L be the side of the smallest square that can contain both the initial configuration and target configuration. Note that, any algorithm requires at least $O(L)$ LCM cycles to solve GMV_{area}. Our algorithm solves GMV_{area} in $O(L)$ LCM cycles which is time optimal. Our result is stated in the following theorem:

Theorem 1. *$\mathcal{A}_{\text{asym}}$ is a time-optimal algorithm that solves GMV_{area} in each asymmetric configuration C defined on a finite grid.*

Proof. Lemmata 1–6 ensure that properties H_1, H_2, and H_3 hold for each task T_{1a}, T_{1b}, …, T_5. Then, all the transitions are those reported in Table 1 and depicted in Fig. 6; the generated configurations can remain in the same task only for a finite number of cycles; and the movements of the robots are all collision-free. Lemmata 1 and 6 also show that from a given task only subsequent tasks can be reached, or $\texttt{pre}_5$ eventually holds (and hence $\mathcal{A}_{\text{asym}}$ is solved). This formally implies that, for each initial configuration C and for each execution $\mathbb{E} : C = C(t_0), C(t_1), C(t_2), \ldots$ of $\mathcal{A}_{\text{asym}}$, there exists a finite time $t_j > 0$ such that $C(t_j)$ is similar to the pattern to be formed in the GMV_{area} problem and $C(t_k) = C(t_j)$ for each time $t_k \geq t_j$.

Concerning the time required by $\mathcal{A}_{\text{asym}}$, it is calculated using the number of required LCM cycles, as robots are synchronous. Recall that L is the side of the smallest square that can contain both the initial configuration and the target configuration. Tasks T_{1a} and T_{1b} require $O(L)$ LCM cycles since a robot must move for $O(\max\{N, M\})$ edges in each of them. Task T_{1c} requires exactly one LCM cycle. By the proof given in Lemma 4, robots complete Task T_2 in at most $2(M-1)$ LCM cycles, that is in $O(L)$ time. Task T_3 requires at most $O(N)$ LCM cycles, i.e. $O(L)$ time. Task T_4 requires exactly one LCM cycle. Then, Algorithm $\mathcal{A}_{\text{asym}}$ requires a total of $O(L)$ LCM cycles, hence it is time optimal since no algorithm can solve GMV_{area} in less than $O(L)$ LCM cycles. □

4.5. The case of symmetric configurations and infinite grids

In this section, we discuss (1) how $\mathcal{A}_{\text{asym}}$ can be extended to a general algorithm $\mathcal{A}$ able to handle also symmetric configurations, and (2) how, in turn, $\mathcal{A}$ can be modified into an algorithm $\mathcal{A}_\infty$ that solves the same problem defined on the infinite grid G_∞.

Symmetric configurations. We first explain how to solve symmetric initial configurations with $\rho(C) = 1$, then those with $\rho(C) \in \{2, 4\}$. If C is a symmetric configuration with $\rho(C) = 1$, then there exists a robot r_c located at the center c of C, and for $C' = \{C \setminus r_c\}$, $\rho(C') \in \{2, 4\}$. To make the configuration asymmetric, $\mathcal{A}$ must move r_c out of c (symmetry-breaking move). To this end, when r_c has an empty neighbor – arbitrarily chosen if more than one – then r_c moves toward it. If all the four neighbors of r_c are occupied but there is at least an empty vertex on the same row or column of r_c, the neighbors of r_c and the robots in front of them until the first empty vertex, move along the row or column. As a result, a neighbor of r_c will eventually be emptied. Then, the symmetry-breaking move can be applied. If all the vertices on the same row and column of r_c are occupied, then all other vertices except one (if any) must be empty. Therefore the four neighbor robots of r_c move toward a vertex placed on the right with respect to c, if empty. Again, a neighbor of r_c will eventually be emptied and the symmetry-breaking move can be applied. When the configuration is made asymmetric, $\mathcal{A}_{\text{asym}}$ runs on C and GMV_{area} is solved.

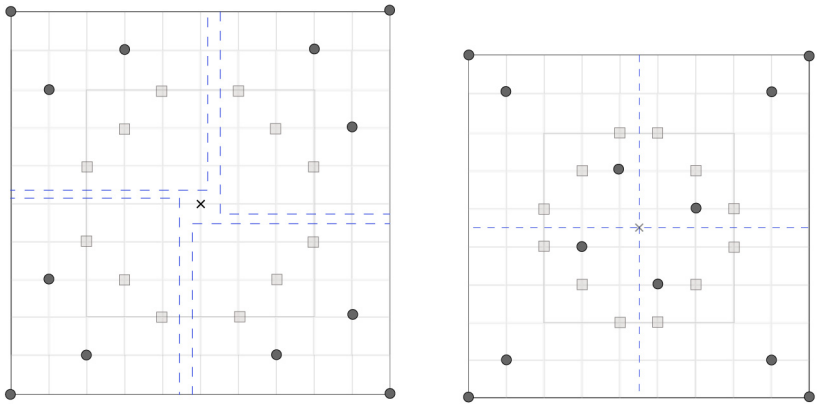


Fig. 7. Patterns F for $\rho(C) = 4$: left $tc(C) = 1$, right $tc(C) = 3$.

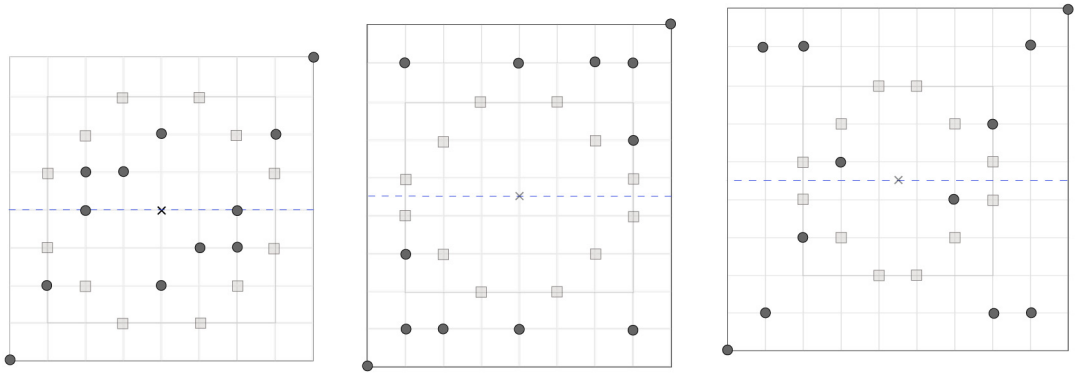


Fig. 8. Patterns F for $\rho(C) = 2$ and $|R| \mod 4 = 0$: left $tc(C) = 1$, middle $tc(C) = 2$, right $tc(C) = 3$.

Consider now C with $\rho(C) \in \{2, 4\}$. In these cases, the configurations is divided into rectangular sectors, i.e., regions of G that are equivalent up to rotations. Then, $\mathcal{A}$ instantiates $\mathcal{A}_{\mathsf{asym}}$ in each sector according to suitably chosen patterns.

We now explain how the subdivision into sectors is performed. Given the symmetry of the configuration, the algorithm $\mathcal{A}$ selects $\rho(C)$ robots as guards and places each of them on different corners of the grid. The placement is done as in $\mathcal{A}_{\mathsf{asym}}$ by means of either tasks T_{1a} and T_{1b} or T_{1c}. Given the placement of the guards, robots identify and enumerate rows and columns, as done in , and agree on how to subdivide G into $\rho(F)$ sectors according to values of $\rho(C)$, of $|R| \mod 4$, and the type of center $tc(C)$.

For configurations having $\rho(C) = 4$ the configuration is divided into four disjoint sectors (cf. Fig. 7): for centers $tc(C) = 1$, each orthogonal line originating from the center is associated to the sector on its left, for centers of type $tc(C) = 3$, sectors are obtained with two orthogonal lines passing through the center of the configuration. When $\rho(C) = 2$ two sectors are obtained with a line parallel to the rows of G passing through c (cf. Figs. 8 and 9). Note that, when $tc(C) = 1$, the line belongs to both sectors. In so doing, sectors keep a rectangular shape and $\mathcal{A}_{\mathsf{asym}}$ can be applied to each of them.

We now explain how patterns are selected and embedded. Figs. 7, 8, and 9 also illustrate some examples concerning the optimal patterns for all cases and for specific values of n. From those examples, patterns F for larger values of n can be easily obtained by suitably enlarging the provided patterns (detailed instructions can be found in [25]). Since robots in C are synchronous, irrespective of the algorithm operating on C, the center c of the configuration is invariant, therefore robots agree on the embedding of F by identifying its center with c and placing the $\rho(F)$ corners of F with the maximum view closest to the $\rho(C)$ guard robots. F is selected so that $\rho(C)$ divides $\rho(F)$ and $tc(F) = tc(C)$, and the placement of robots in F solves $\mathrm{GMV}_{\mathsf{area}}$.

As pointed out before, each sector contains a sub-configuration that is asymmetric, then $\mathcal{A}$ instantiates $\mathcal{A}_{\mathsf{asym}}$ in each sector while the definitions of functions A_l, B_l, RD_l, and RU_l apply to each sector. Note that the algorithm works correctly even for configurations having $\rho(C) = 2$ and $tc(C) = 1$ where the two sub-grids, in which the $\mathcal{A}_{\mathsf{asym}}$ runs independently, share the central row of G. In particular, the number of robots and targets is computed by each instance of $\mathcal{A}_{\mathsf{asym}}$ only considering those lying on the half of the central row closest to the guard. Note that, the center is never considered by the computation as there is neither a target nor a single robot there. If a robot from a sector, say the first one, moves on

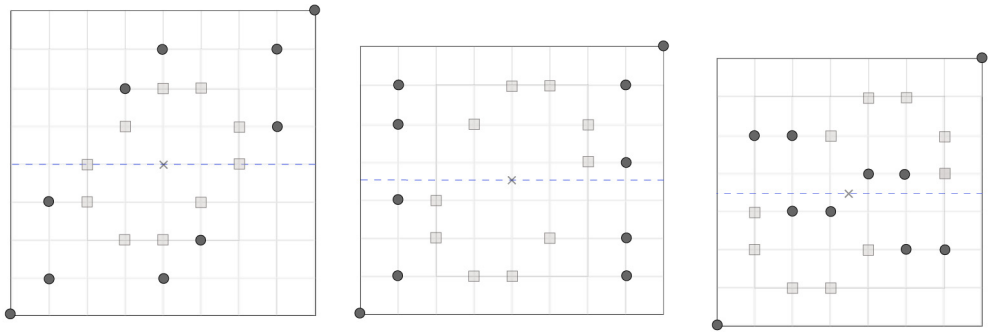


Fig. 9. Patterns F for $\rho(C) = 2$ and $|R| \mod 4 = 2$: left $tc(C) = 1$, middle $tc(C) = 2$, right $tc(C) = 3$.

the central row, it may fall into the half row belonging to the other sector, say the second one, but its equivalent robot in the second sector would move in the opposite direction entering the half row belonging to the first sector and the number of robots in each sector is kept. In such situations, two robots may move and collide on the center of C, $c(R)$. In this case, we need to slightly modify $\mathcal{A}_{\mathsf{asym}}$: robots are prevented from moving on $c(R)$ while other robots will eventually move on the central row.

Another difference with $\mathcal{A}_{\mathsf{asym}}$, is the movement of the guard robot toward its final target in F during task T_4. In this case, r_g is not one step away from its target as in the asymmetric case, given the embedding of F into the center of G. Move m_4 works also in $\mathcal{A}$, but it is completed in more than one LCM cycle.

Infinite grids. To obtain $\mathcal{A}_\infty$, it is sufficient to make small changes to tasks T_{1a}, T_{1b}, and T_4. In $\mathcal{A}_{\mathsf{asym}}$, task T_{1a} selects a single robot r_g to occupy a corner of G. Since G_∞ does not have corners, $\mathcal{A}_\infty$ selects r_g as in T_{1a} and then moves it to a distance $\mathcal{D} \geq 3 \cdot \max\{w(C'), w(F)\}$, where $C' = \{C \setminus r_g\}$, and $w(C')$, $w(F)$ are the longest sides of $mbr(C')$ and $mbr(F)$, respectively. In task T_{1b}, r_g must be chosen as the robot with a distance $\mathcal{D}$ from C', and it moves toward a corner of C. In T_2, the first row is identified as the first row of C' occupied by a robot, approaching C' from r_g. The embedding on F is achieved by matching the corner of F with the maximum view in correspondence with the corner of C' on the first row and having the same column of r_g. Tasks T_2 and T_3 are unchanged, while in task T_4, r_g takes $\mathcal{D}$ LCM cycles to move toward its final target in F. Because of the specific pattern F, the detection of r_g will be always guaranteed being such a robot the only one that can finalize the formation of F with a straight movement toward its target.

5. Solving GMV on trees

In this section, we address GMV for robots moving on trees. We first provide all the necessary concepts to define what we call "critical vertices", and then we restrict the problem to configurations that do not contain these kinds of vertices. The restricted version of the problem, denoted as GMV$_{\bar{c}}$, is then solved by devising an algorithm that works for semi-synchronous robots and assumes the weak robot model described in Section 2.

5.1. Critical vertices and the addressed problem

Given a configuration $C = (T, \lambda)$, a vertex v of T such that $\lambda(v) > 0$ is called *occupied*, *unoccupied* otherwise. If v is occupied by a robot r, we often denote v also as v_r. Notation $\ell(T)$ is used to represent the number of leaves of T. As usual, $N(v)$ denotes the set of all adjacent vertices of a vertex v. Assuming $N(v) = \{v_1, v_2 \ldots, v_k\}$, the removal of v from T creates k subtrees, each denoted as $T(v, v_i)$ and assumed rooted at v_i, $i = 1, 2, \ldots, k$. Given two nodes v_1 and v_2 of T, if $e = (v_1, v_2)$ is an edge of T, the removal of e from T creates two subtrees, each denoted as $T(e, v_i)$ and assumed rooted at v_i, $i = 1, 2$. In each removal operation, the obtained subtrees are called *complete subtrees* of T. These removal operations are now used to provide some additional definitions.

Definition 1. Let $C = (T, \lambda)$ be a configuration, and T' be a complete subtree of T obtained by a removal operation. T' is *overloaded* if the number of robots in T' is greater than the number of leaves of T in T'.

Fig. 10 shows three configurations where the complete subtrees of vertex v with robots are overloaded subtrees.

Definition 2. Let $C = (T, \lambda)$ be a configuration. An edge $e = (v_1, v_2)$ of T is considered *oriented* from v_1 to v_2 if the complete subtree $T(e, v_1)$ is overloaded. A path $P = (v_1, v_2, \ldots, v_k)$ of T is considered *oriented* if all its edges are oriented toward the same endpoint, i.e., either v_1 or v_k. P is considered *partially-oriented* if all the oriented edges (if any) are oriented toward the same endpoint.

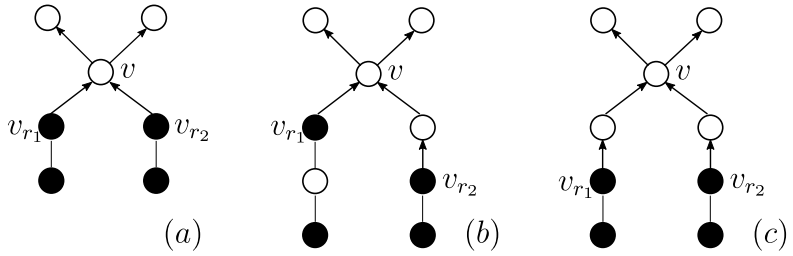


Fig. 10. Examples of configurations. Occupied vertices are represented in black.

Consider again Fig. 10. All the given configurations are represented according to the edge orientation described in the above definition. In each case, the path from v_{r_1} to any unoccupied leaf is oriented, whereas the path from any occupied leaf to any unoccupied leaf is partially-oriented.

Definition 3. Let $C = (T, \lambda)$ be a configuration, and v be a vertex of T. Vertex v is *critical* if its removal generates at least two overloaded complete subtrees $T(v, v_1)$ and $T(v, v_2)$ such that $(T(v, v_1), \lambda)$ and $(T(v, v_2), \lambda)$ are isomorphic. In such a case, these subtrees are called *critical-subtrees*. Vertex v is *potentially-critical* if its removal generates at least two overloaded complete subtrees and all such generated subtrees are pairwise non-isomorphic.

As an example, vertex v in the configuration given in Fig. 10.(a) is critical (in fact, $(T(v, v_{r_1}), \lambda)$ and $(T(v, v_{r_2}), \lambda)$ are isomorphic and both overloaded). In Fig. 10.(b), instead, vertex v is potentially-critical as it is non-critical but the two sub-trees below it are both overloaded. The term potentially-critical is motivated by the observation that when moving robots from an overloaded subtree toward unoccupied leaves, the performed move could transform the vertex from potentially-critical to critical (and this, as it will be clarified in Section 5.7, could generate unsolvable configurations).

The addressed problem. The specific version of GMV addressed in this section is denoted as $\text{GMV}_{\overline{c}}$ and is defined as follows: given a configuration $C = (T, \lambda)$ without critical vertices and with $n \leq \ell(T)$ robots located on n distinct vertices, design a deterministic distributed algorithm that transforms C into a configuration $C' = (T, \lambda')$ in which each robot occupies a distinct leaf of T. Notice that, C' has all robots positioned on the leaves, and this ensures that $\text{GMV}_{\overline{c}}$ is solved even if the original problem definition does not require such a property.

In the remainder of this section, we will provide an algorithm that is able to solve $\text{GMV}_{\overline{c}}$. We first show how the algorithm works in the extreme case with $n = \ell(T)$ robots. Successively, we give the necessary modifications for the general case with $n \leq \ell(T)$.

We remark that, in general, the solvability of many algorithmic problems defined for robots moving in a discrete or continuous environment is strongly influenced by symmetries in the input configuration, and therefore by the presence of pairwise equivalent robots. It is important to note that we do consider symmetric configurations, but without critical vertices. In Section 5.7, we provide an extensive discussion in which we motivate how the presence of critical vertices makes solving GMV on trees particularly difficult, if not even impossible.

5.2. Further notation and definitions

Given a configuration $C = (T, \lambda)$, we denote by $R = \{r_1, r_2, \ldots, r_{\ell(T)}\}$ the set of robots in C.[2] The *center* of a graph is the set of all vertices that minimize the maximal distance from other points in the graph. It is well known that the center of a tree is a set containing one vertex or two adjacent vertices [33]. The provided algorithm requires that each robot identifies a single vertex as center, denoted as $c(T)$.[3] To this aim, when the center of T is a single vertex v, then each robot assumes $c(T) = v$; when the center is $\{v_1, v_2\}$ and $e = (v_1, v_2)$ is oriented toward v_i, then each robot assumes $c(T) = v_i$; when the center is $\{v_1, v_2\}$ and $e = (v_1, v_2)$ is non-oriented, each robot located in the subtree $T(e, v_i)$ assumes $T = T(e, v_i)$ and $c(T) = v_i$, $i = 1, 2$, that is like running the algorithm concurrently in two distinct instances.

We denote by $\mathcal{P}(C)$ the set containing all the partially-oriented paths from $c(T)$ to some unoccupied leaf of T, if any. We will show that $\mathcal{P}(C) \neq \emptyset$ for each configuration C where $\text{GMV}_{\overline{c}}$ is not yet solved.

Definition 4. Let $C = (T, \lambda)$ be configuration without critical vertices and multiplicities. $R'(C)$ is the set containing any robot $r \in R$ such that:

[2] We recall that we are first considering the extreme case of $n = \ell(T)$ robots and that the robots are anonymous. The notation is used only for the sake of presentation, since robots are anonymous no algorithm can take advantage of names of elements in R.

[3] Although there is a little abuse in the notation, the definition of $c(T)$ does not have to be confused with that provided in Section 4 referring to the geometric center of a grid.

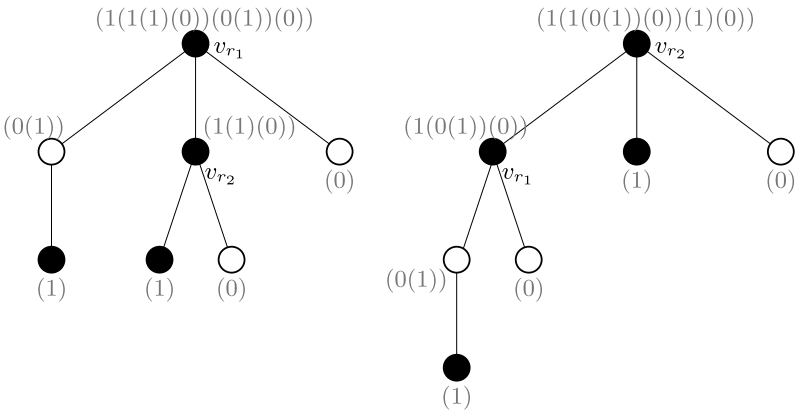


Fig. 11. Examples of views associated with different robots/vertices.

- r is on a vertex of a path $P \in \mathcal{P}(C)$ leading to an unoccupied leaf l;
- r is the closest robot to l among the robots on vertices of P;
- the subpath of P is oriented from v_r to l.

For each edge $e = (u, v)$ of T, let $s(e)$ be the minimum number of robots that have to pass through e to solve $\text{GMV}_{\bar{c}}$ on C. Formally, if e is not oriented, then $s(e) = 0$; if e is oriented from u to v, then $s(e)$ is the difference between the number of robots on the vertices of $T(e, u)$ and the number of leaves of T in $T(e, u)$. For a partially-oriented path P, $s(P) = \sum_{e \in P} s(e)$.

View of a robot. In the algorithm we provide for solving $\text{GMV}_{\bar{c}}$, sometimes we need to distinguish among robots having some properties (e.g., minimum distance from unoccupied leaves). To this purpose, to model the *view* of a robot, we consider an approach similar to that used in [34,35] to determine isomorphisms among trees. In particular, a robot r can associate a unique string to the tree rooted in the vertex v_r where it resides, keeping trace of the presence/absence of a robot in a vertex by associating 1 or 0, respectively. Moreover, parentheses are inserted into the strings to track the relationship between one node and its children recursively. For example, the string associated with the vertex v_{r_1} in Fig. 11 is (1(1(1)(0))(0(1))(0)), obtained by lexicographically ordering the strings recursively associated with the roots of its subtrees. The lexicographic order assumes "(" < ")" < "1" < "0". Since the string associated with v_{r_2} is (1(1(0(1))(0))(1)(0)), then we say that the view of robot r_1 is smaller than the view of robot r_2. Notice that two equivalent robots have the same view (i.e., are associated with the same string). In conclusion, each robot can compute the view of all robots, determine the robot(s) with minimum view, and also determine whether there is any symmetry in the observed configuration.

5.3. Description of the algorithm

The provided algorithm for solving $\text{GMV}_{\bar{c}}$ is called `MoveToLeaf` and it has been designed according to the methodology recalled in Section 3.2. The algorithm is based on three tasks named T_{ep}, T_{fp}, and T_t: T_{ep} is responsible for "emptying paths in $\mathcal{P}(C)$"; T_{fp} for "filling paths in $\mathcal{P}(C)$"; T_t for checking the "termination", that is checking that $\text{GMV}_{\bar{c}}$ is solved and no further move is necessary.

According to the reduced number of tasks, here the algorithm is formalized in pseudo-code instead of the tabular form as used in Section 4. The pseudo-code is described in Algorithm 4. Essentially, we can assume that, during a `LCM`-cycle, each robot first acquires a snapshot of the current configuration (in the `Look` phase), and then executes `MoveToLeaf` (in the `Compute` phase). In the `Move` phase, the moving robot performs the move as specified in `MoveToLeaf`.

To check which task must be performed, the algorithm uses simple predicates based on $R'(C)$ and $\mathcal{S}$, where the latter is a data structure computed by the associated Procedure `DetermineMovingRobots` (cf. the pseudo-code described in Algorithm 5). It can be easily observed that the used predicates are well-formed since both $R'(C)$ and $\mathcal{S}$ fulfill properties Prop_1, Prop_2, and Prop_3 defined in Section 3.2.

The strategy behind algorithm `MoveToLeaf` is the following. At line 2, the set $R'(C)$ is computed. If such a set is not empty, then there are robots on partially-oriented paths from $c(T)$ toward unoccupied leaves which can be brought to target by moving them along oriented paths. This case can be observed in Fig. 12, where robots r_1 and r_2 belong to $R'(C)$. In this situation, the algorithm performs task T_{ep}: it preliminarily moves these robots (cf. move m_1 at Line 5) until a configuration C_1 in which $R'(C_1) = \emptyset$ is generated.

Successively, at Line 7, since $R'(C)$ is empty, `MoveToLeaf` calls procedure `DetermineMovingRobots`. Assume that a robot r (located on some non-leaf vertex v_r) can move along an oriented path P to reach a vertex v that belongs to any partially-oriented path in $\mathcal{P}(C)$. `DetermineMovingRobots` associates a priority to r according to the integers $s(e)$ assigned to each edge e of P (an example of this assignment is shown in Fig. 12). In this way, a sequence of integers is

Algorithm 4 `MoveToLeaf`

Input: A configuration $C = (T, \lambda)$ without critical vertices and multiplicities.
compute the view of C
compute $R'(C)$
if $R'(C) \neq \emptyset$ **then**
  {task T_{ep}}
  move m_1*:* each robot $r \in R'(C)$ of minimum view moves toward one of its closest unoccupied leaves along a path $P \in \mathcal{P}(C)$ toward one of such leaves
else
  let $\mathcal{S} =$ `DetermineMovingRobots`(C)
  if $\mathcal{S} \neq \emptyset$ **then**
    {task T_{fp}}
    move m_2*:* let (v, v_r) be the entry of $\mathcal{S}$ with r of minimum view, r moves toward v
  else
    {task T_t}
    move m_3*: nil*

Algorithm 5 `DetermineMovingRobots`

Input: A configuration $C = (T, \lambda)$ without critical vertices and multiplicities.
let $\mathcal{S}$ be an empty map
compute $\mathcal{P}(C)$
for all $P \in \mathcal{P}(C)$ **do**
  let l be the leaf where P leads
  let v be the vertex on P closest to l such that there exists an edge $e = (u, v)$ oriented toward v with u not in P
  for all occupied vertex $v_r \in T(v, u)$ **do**
    if the path $P(v, v_r) = (v \equiv v_0, v_1, v_2, \ldots, v_t \equiv v_r)$ is oriented toward v **then**
      let $\mathcal{S}[(v, v_r)] = (s((v_0, v_1)), s((v_1, v_2)), \ldots, s((v_{t-1}, v_r))$
let $\mathcal{S}'$ be the submap of $\mathcal{S}$ containing the lexicographically minimal sequences of $\mathcal{S}$
return $\mathcal{S}'$

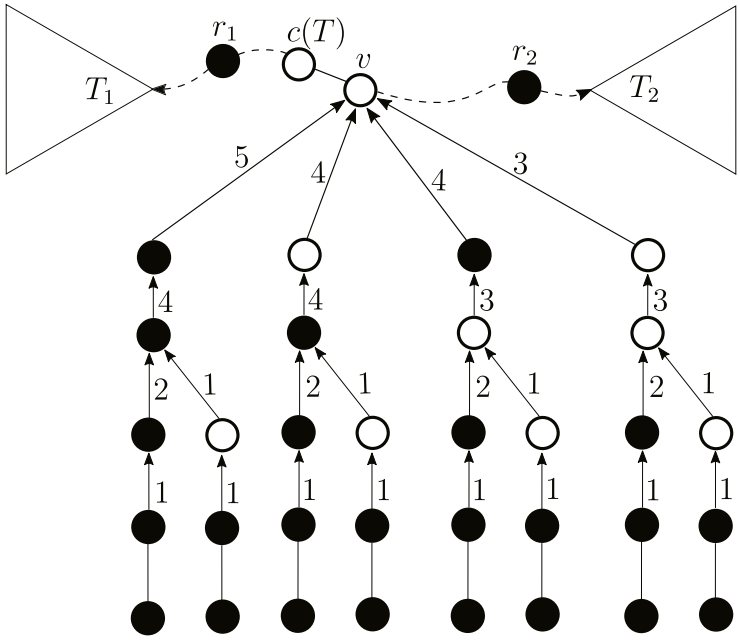


Fig. 12. A schematic representation of a configuration C elaborated by `MoveToLeaf`. The triangles represent two subtrees denoted as T_1 and T_2 and containing unoccupied leaves. The dashed and curved lines represent paths. In the discussion, it is assumed that the paths from $c(T)$ to T_1 and from $c(T)$ to T_2 belong to $\mathcal{P}(C)$.

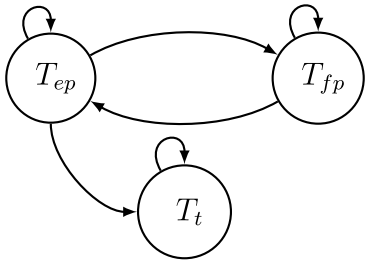


Fig. 13. Transition graph associated with Algorithm `MoveToLeaf`.

assigned to robots, and the robot with the lexicographically smallest sequence is then moved by `MoveToLeaf` along an oriented path toward a path in $\mathcal{P}(C)$. Notice that, if the configuration has vertices equivalent to v, equivalent robots can be selected and moved concurrently (but remember that their activation is decided by the adversary). We remark that the priority based on the integer sequences is an essential part of the strategy as it avoids performing "bad moves" that could transform vertices from potentially-critical to critical thus generating unsolvable configurations. For instance, the rightmost sequence represented in Fig. 12, it can be observed that the lexicographically smallest sequence (3, 3, 1, 1) is associated to the only robot that can reach v without creating critical vertices and following a path in $\mathcal{P}(C)$.

By considering again the current scenario, it follows that `MoveToLeaf` calls `DetermineMovingRobots` to select one robot (and its equivalent robots) to be moved toward a partially-oriented path from $c(T)$ to unoccupied leaves (and this corresponds to task T_{fp}). When this path is reached by a robot, set R' turns out to be not empty and hence move m_1 is applied again to lead that robot on an unoccupied leaf. The whole process is repeated until a final configuration is created. In that case, we have $R'(C) = \emptyset$ and $\mathcal{S} = \emptyset$ and hence robots detect that task T_t is running and no further moves are necessary. The transitions among the three tasks are represented in Fig. 13.

5.4. Correctness

We have already remarked that `MoveToLeaf` has been designed according to three tasks only and that the used predicates are all well-formed. The corresponding transition graph is depicted in Fig. 13. Concerning the correctness, in what follows we provide a sequence of lemmata that are either structural (e.g., showing properties about $\mathcal{P}(C)$) or show properties about the algorithm once a single execution of m_1 (made in task T_{ep}) or m_2 (made in task T_{fp}) occurs. Finally, such lemmata are exploited by a theorem that provides the correctness of `MoveToLeaf`. In particular, as introduced in Section 3.2, our algorithm needs to fulfill conditions $H_1 - H_3$. Condition H_3 is implied by Lemmata 8 and 10. Whereas conditions H_1 and H_2 are verified in Theorem 2.

Lemma 7. *Let $C = (T, \lambda)$ be a configuration without critical vertices and multiplicities. If $\text{GMV}_{\bar{c}}$ is not solved in C, then $\mathcal{P}(C) \neq \emptyset$.*

Proof. Since $\text{GMV}_{\bar{c}}$ is not solved in C, there exists at least one unoccupied leaf. By contradiction, assume $\mathcal{P}(C) = \emptyset$. This implies that each path from $c(T)$ to an unoccupied leaf has at least one edge oriented toward $c(T)$. Let P_1 be one of such paths and let $e = (v_1, v_2)$ be an edge of P_1 oriented toward $c(T)$. Removing e from T generates the subtrees $T(e, v_1)$ and $T(e, v_2)$. Without loss of generality assume that $c(T)$ is contained in $T(e, v_1)$. By definition of oriented edge, in $T(e, v_1)$ the number of robots is strictly less than the number of leaves of T in $T(e, v_1)$. Hence $T(e, v_1)$ contains at least an unoccupied leaf l. Let P_2 be the path from $c(T)$ to l. Let then remove one edge oriented toward $c(T)$ from P_2 and consider again the generated subtree containing $c(T)$. Repeat this procedure until the generated subtree T^* containing $c(T)$ has no unoccupied leaf. In T^*, the number of robots is strictly less than the number of leaves of T in T^*, but the number of unoccupied leaves of T^* is zero, a contradiction. □

Lemma 8. *Let $C = (T, \lambda)$ be a configuration without critical vertices and multiplicities, and let C' be the configuration generated from C by `MoveToLeaf` according to one execution of move m_1. Then, C' contains neither critical vertices nor multiplicities.*

Proof. Consider the set R'' containing all the equivalent robots moved by move m_1. We have to show that the configuration C', created after the move of the robots in R'', contains neither critical vertices nor multiplicities. By Lemma 7 and definition of $R'(C)$, each robot in R'' admits a distinct directed path toward an unoccupied leaf where no other robots lie. Hence, the creation of multiplicities along such paths is not possible. Let $r \in R''$ and, by contradiction, let u be a critical vertex generated after the move of r and the robots equivalent to r in R'', if any. Vertex u must be on the path from r to $c(T)$, otherwise it was a critical vertex even before the move. Since u is critical, there must be two or more pairwise isomorphic subtrees created after the move of r. Robot r must be in one of them, say $T(u, v)$. Since $T(u, v)$ is overloaded, the edge

(u, v) must be oriented from v to u. This is a contradiction since (u, v) belongs to the path from $c(T)$ to the unoccupied leaf l, target of r, and this path is partially-oriented from $c(T)$ to l. □

Lemma 9. *Let $C = (T, \lambda)$ be a configuration without critical vertices and multiplicities. If $R'(C) = \emptyset$ then each $P \in \mathcal{P}(C)$ does not contain any occupied vertex.*

Proof. By contradiction, assume that there exists a path $P \in \mathcal{P}(C)$, partially-oriented from $c(T)$ to an unoccupied leaf l, with some occupied vertices. Let r be the robot on P closest to l. Denote as P' and P'' the subpaths of P from $c(T)$ to v_r and from v_r to l, respectively. According to the definition of $R'(C)$, the assumption $R'(C) = \emptyset$ implies that P'' is not oriented toward l. Since the number of robots is equal to $\ell(T)$, there must exist an oriented path P''' from v_r to an unoccupied leaf $l' \neq l$. Since P' is partially-oriented toward v_r, then P' and P''' do not share any edge. Hence, the concatenation of P' and P''' forms a partially-oriented path from $c(T)$ to l'. A robot in this path (either r or the robot in the path that is closest to l') fulfills Definition 4. Hence $R'(C) \neq \emptyset$, against the assumption. □

Lemma 10. *Let $C = (T, \lambda)$ be a configuration without critical vertices and multiplicities, and let C' be the configuration generated from C by* `MoveToLeaf` *according to one execution of move m_2. Then, also C' contains neither critical vertices nor multiplicities.*

Proof. Since `MoveToLeaf` applies move m_2 then $R'(C) = \emptyset$. By Lemma 9, each path $P \in \mathcal{P}(C)$ does not contain robots. Consider the set R'' containing all the equivalent robots moved by move m_2. We have to show that the configuration C', created after the move of the robots in R'', contains neither critical vertices nor multiplicities.

Move m_2 selects a pair (v, v_r) and moves the robot r toward v. The algorithm moves all the robots equivalent to r, and then with minimal view, if any. When r is moving toward v, say from v_r to $v' \in N(v_r)$, there are two cases in which a critical vertex can be created:

1. the complete subtree $T(v', v_r)$ becomes isomorphic to another tree $T(v', a)$, hence v' becomes critical;
2. robot r becomes equivalent to a robot r'.

(Case 1) Before r moves, $T(v', v_r)$ has one robot more than $T(v', a)$. Notice that there must be at least one robot in both the sub-trees. Hence, $s((v', v_r)) > s((v', a))$ and then $\mathcal{S}[(v, v_r)] > \mathcal{S}[(v, a)]$. This implies that a robot in $T(v', a)$ had to be moved instead of r.

(Case 2) After the move of r on v', a new critical vertex u is created at the center of the path Q between v' and $v_{r'}$, with the two incident edges on paths $P(v', u)$ and $P(v_{r'}, u)$ oriented toward u. Moreover, u is the vertex closest to $c(T)$ among the vertices in Q. Then u is in the path $P(c(T), v)$, subpath of P, or in the path $P(v, v')$. Vertex u cannot be a vertex of $P(c(T), v)$, v excluded, due to the orientation of the edges of P toward an unoccupied leaf. Then, u is a vertex in the path $P(v, v')$ (extremes included). Since robots r and r' are equivalent, we have $s(P(v, v')) = s(P(v, v_{r'}))$. Then, $s(P(v, v_{r'}))$ is a prefix of $s(P(v, v_r))$ before the move. So, $s(P(v, v_{r'})) < s(P(v, v_r))$ and the robot to be moved was r', indeed.

As for the multiplicities, if r is the only robot moving on $v' \neq v$ then no multiplicity is possible since, by the minimality of $s(P(v, v_r))$, v' is unoccupied. If r is the only robot moving on v' when $v' = v$, then v' is unoccupied because it is on a path $P \in \mathcal{P}(C)$ and, since $R'(C) = \emptyset$, by Lemma 9, P has no occupied vertices. If r is not the only robot moving on v', then all the robots moving on v' are equivalent, and v' is critical in C, a contradiction to the assumption that C does not contain critical vertices. □

Theorem 2. *Let $C = (T, \lambda)$ be a configuration composed of $n = \ell(T)$* SSync *robots. If C contains neither critical vertices nor multiplicities, then* `MoveToLeaf` *correctly solves* $\text{GMV}_{\vec{c}}$ *from C.*

Proof. Given $s(C) = \sum_{e \in E} s(e)$, it easily follows that GMV is solved from C if and only if $s(C) = 0$. Let $\mathbb{E} : C = C(0), C(1), C(2), \ldots$ be an execution of `MoveToLeaf` formed by a sequence of configurations observed at discrete time $t = 0, 1, 2, \ldots$. We have to show that there exists $t^* \geq 0$ such that $C(t^*) \in \mathbb{E}$, $s(C(t^*)) = 0$, and $C(t) = C(t^*)$ for each $t > t^*$.

Assume $s(C(0)) > 0$, and without loss of generality $R'(C(0)) \neq \emptyset$. This implies that `MoveToLeaf` performs task T_{ep} by applying m_1 to $C(0)$.

By Lemma 8 the resulting configuration $C(1)$ is still without critical vertices and without multiplicities. Moreover, $s(C(1)) < s(C(0))$ because m_1 moves robots toward unoccupied leaves along oriented edges. Hence, by repeatedly applying m_1, `MoveToLeaf` leads to a configuration $C(t')$, $t' > 0$, without critical vertices and multiplicities and such that $R'(C(t')) = \emptyset$. In $C(t')$, `MoveToLeaf` performs task T_{fp} by applying move m_2. Lemma 10 guarantees that $C(t' + 1)$ is still without critical vertices and without multiplicities. In particular: (1) move m_2 moves robot r (along with its equivalent robots), (2) v_r belongs to a path $P(v, v_r)$ oriented from v_r to v, and (3) r moves along $P(v, v_r)$ toward v. This implies that $s(C(t' + 1)) < s(C(t'))$. If in $C(t' + 1)$ robot r does not reach v, move m_2 is applied again. Assume that at time t'', for some $t'' > t' + 1$, r reaches v. Since v is on a path $P \in \mathcal{P}(C(t''))$, by Lemma 9 we get $R'(C(t'')) \neq \emptyset$ and move m_1 is applied again.

As depicted in the transition graph shown in Fig. 13, it follows that the execution $\mathbb{E}$ is formed by alternating subsequences of configurations in which each subsequence is generated only by move m_1 or only by move m_2. Since we have shown that the function $s()$ decreases at each execution of `MoveToLeaf`, it is clear that condition H_2 is fulfilled, that is there exists a time $t^* > 0$ such that $s(C(t^*)) = 0$, $R(C(t^*)) = \emptyset$, and the map $\mathcal{S}$ is empty. Hence, robots detect that task T_t is reached in $C(t^*)$ and hence no further moves are made. By the above discussion, it is clear that also condition H_1 (the transitions between tasks are exactly those represented in the transition graph) is fulfilled and this means that `MoveToLeaf` is able to solve GMV$_{\bar{c}}$ from C. □

5.5. Case with $n \leq \ell(t)$ robots

So far, the proposed algorithm has been designed to approach the extreme case with $n = \ell(T)$ robots. In this section, we provide all the details necessary to deal also with $n < \ell(T)$ robots. In general, the strategy will be to add $\ell(T) - n$ virtual (and static) robots to the configuration so as to allow the use of the algorithms described before. In particular, the added virtual robots are used to compute all the directions on the edges of the input tree in order to define the set of paths $\mathcal{P}(C)$. Actually, virtual robots are not used to compute $R'(C)$ or any other subset involving robots. Of course, for consistency reasons, all robots must agree about the same locations where to add the virtual robots. Moreover, we have to guarantee that such robots do not affect the normal functioning of the algorithms designed for the case of $n = \ell(T)$.

About the location(s) where to add $\ell(T) - n$ virtual robots, we consider the center of the input tree T. In particular, if the center of T is a set containing just one vertex $c(T)$, then robots can compute the directions of the edges by adding $\ell(T) - n$ virtual robots in $c(T)$. When the center is $\{v_1, v_2\}$, we remind that there exists the edge $e = (v_1, v_2)$. Let n_i be the number of robots residing in the subtree $T(e, v_i)$, $i = 1, 2$. Now, if v_i is not a leaf of $T(e, v_i)$ then set $\rho_i = \ell(T(e, v_i)) - n_i$, otherwise set $\rho_i = (\ell(T(e, v_i)) - 1) - n_i$. If $\rho_i > 0$, then add ρ_i virtual robots to v_i, for $i = 1, 2$. In doing so, we ensure that the role of the virtual robots never changes, as well as their positioning.

Let us now consider the functioning of the proposed algorithms. First of all, Algorithm `DetermineMovingRobots` is not affected by virtual robots as by construction $c(T)$ is never part of the subtree considered by that algorithm. Concerning Algorithm `MoveToLeaf`, instead, it would move robots from $c(T)$ if the paths to the leaves do not contain robots. Since virtual robots are not accounted in $R'(C)$, the algorithm would not allow virtual robots to move. Hence, if Algorithm `MoveToLeaf` has still robots to move, it proceeds; otherwise, if only virtual robots remain to move then it means GMV$_{\bar{c}}$ has been solved.

By Theorem 2 and the above discussion, we conclude the following final statement.

Theorem 3. *Let $C = (T, \lambda)$ be a configuration without critical vertices and composed of $n \leq \ell(T)$ SSync robots. Then, `MoveToLeaf` correctly solves GMV$_{\bar{c}}$ from C.*

5.6. Time complexity

The time complexity is measured in terms of *epochs*, where an epoch is the time duration for all robots to execute at least one complete `LCM`-cycle since the end of the previous epoch. For FSync robots, an epoch coincides with a round. For SSync and Async robots, instead, the duration of an epoch may vary from time to time and it is unknown, however, by the fairness condition, it is finite.

In what follows, D denotes the diameter of the tree of a given configuration.

Lemma 11. *`MoveToLeaf` requires $O(nD)$ epochs to solve GMV$_{\bar{c}}$ from configurations without critical vertices and composed of n SSync robots.*

Proof. The statement simply follows by observing that procedure `MoveToLeaf`, at Line 5 or at Line 10, always moves a robot at a time (along with all the robots equivalent to it) along shortest oriented paths toward a target leaf which might be of length D. □

Lemma 12. *GMV$_{\bar{c}}$ requires $\Omega(n + D)$ epochs to be solved from configurations without critical vertices and composed of n SSync robots.*

Proof. Let $C = (T, \lambda)$ be a configuration without critical vertices and composed of n SSync robots $r_1, r_2, \ldots, r_n$. Assume that T is a tree consisting of a path $P = (v_1, v_2, \ldots, v_n, \ldots, v_m)$ such that $m \gg n$, along with $n - 1$ pendant vertices attached to v_m. Assume $r_1, r_2, \ldots, r_n$ are disposed on $v_1, v_2, \ldots, v_n$, respectively. This implies that r_1 is already on a target, whereas the remaining $n - 1$ robots must be moved on the $n - 1$ leaves connected to v_m. In $C = C(0)$, only v_n can be moved, otherwise a collision is created. Let $C(1)$ be the configuration obtained after the move of v_n. In $C(1)$, only robots v_n and v_{n-1} can be moved. By repeating this analysis, we get that each solving algorithm can move v_2 for the first time no earlier than $t = n - 2$. After this first movement, v_2 requires $D - 1$ additional epochs to reach its target. At that time, GMV$_{\bar{c}}$ is solved. This implies that any solving algorithm requires $\Omega(n + D)$ epochs to solve GMV$_{\bar{c}}$ on the assumed input configuration. □

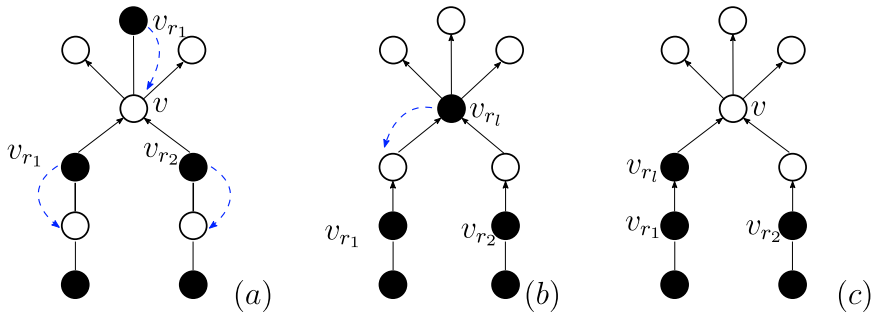


Fig. 14. (a): A configuration with a critical vertex v. The dashed arrows show the direction proposed for the movement of the robots in order to solve GMV; (b): the symmetric configuration obtained after the one on the left with a new proposed movement; (c): the configuration made asymmetric.

5.7. On the difficulties posed by critical vertices

In this section, we illustrate some of the challenges posed by GMV on trees when critical vertices are allowed. To this aim, we show a few cases of input configurations with critical vertices that are either unsolvable or require specific strategies within SSync. Furthermore, we discuss how they can be approached within FSync.

1. Unsolvable configurations. Consider the configuration C_1 shown in Fig. 10.(a). Vertex v is critical in C_1 since $(T(v, v_{r_1}), \lambda)$ and $(T(v, v_{r_2}), \lambda)$ are isomorphic and both overloaded (in particular, each vertex in these critical-subtrees is occupied). Notice that, in C_1 each resolving algorithm for GMV should move robots r_1 and r_2 toward v. However, since r_1 and r_2 are equivalent, no algorithm can distinguish between the two subtrees and decide which robot among r_1 and r_2 should make a step toward the parent vertex v. Hence, each algorithm would create a collision in v. Since the definition of GMV requires to not incur in collisions, then GMV results to be unsolvable in C_1, as stated in the following claim.

Claim 1. *Let C be a configuration without multiplicities. Assume also that in C there is a critical vertex admitting critical-subtrees having all vertices occupied. Then,* GMV *cannot be solved from C even by* FSync *robots.*

In fact, when the conditions of this claim hold, it is clear that if a robot enters (exits from or moves within) any of the critical-subtrees then a multiplicity is created.

Fig. 10.(c) shows another unsolvable configuration in which the previous claim does not apply. Theoretically, if other robots are present, in some cases it is possible to move them inside critical-subtrees so as to break the symmetry and solve the problem. GMV cannot be solved from the configuration shown in Fig. 10.(c) because there are no robots that can be used to break the symmetry. The following paragraph presents a deeper analysis.

2. Using leader robots to remove critical vertices.

Consider the configuration shown in Fig. 14.(a). The vertex v is critical since it has two subtrees, $T(v, v_{r_1})$ and $T(v, v_{r_2})$ that are isomorphic and overloaded. As in the previous case, it can be observed that r_1 and r_2 cannot move directly on v otherwise they would collide. By observing that $T(v, v_{r_1})$ and $T(v, v_{r_2})$ are not completely occupied – as it happens in Fig. 10.(a), a resolving strategy could move a robot inside one of the two isomorphic subtrees in order to break the symmetry and hence transforming v from critical to potentially-critical.

In fact, the robot r_l (which can be elected as a "leader" since it has no equivalent robots) can actually move toward v while r_1 and r_2 move downward. As we are in SSync (but the approach seems to be effective also in Async), such moves do not necessarily happen concurrently. In particular, if for instance r_1 moves before r_2, then the algorithm may let robots wait for r_2 to move.

After that, as in Fig. 14.(b), r_l can move toward one of the two symmetric subtrees, hence breaking the symmetry, and thus obtaining the configuration in Fig. 14.(c). In so doing, the critical vertex v actually becomes potentially-critical. From there, r_2 can freely move toward a leaf, and afterward r_l and r_1, in turn, can reach their destination leaves, solving GMV.

Notice that the proposed strategy clearly requires (1) the presence of a leader robot outside the two isomorphic subtrees, and (2) that the involved isomorphic subtrees admit at least one unoccupied vertex where the leader robot can enter to break the symmetry. This leads to the following claim:

Claim 2. *Let C be a configuration without multiplicities in which there is a critical vertex but no leader robots. Then,* GMV *cannot be solved from C even by* FSync *robots.*

Fig. 15.(a) shows a more general case with respect to Fig. 14 since v has more than two isomorphic critical-subtrees. A resolution strategy moves a leader robot r_l inside one of the critical-subtrees see Fig. 15.(a) and then the algorithm creates a configuration on other critical-subtrees so that to be different from any other configuration created on the first subtree

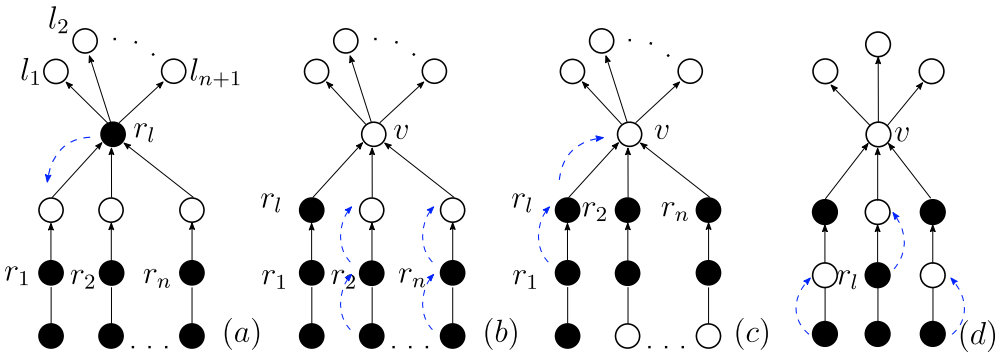


Fig. 15. (a): A symmetric configuration with more than two isomorphic overloaded subtrees. Vertex v is critical, r_l is a leader robot, r_l moves inside the first subtree (b): robots of other subtrees, move toward v, (c): the first subtree can be emptied. (d): v is critical and r_l is the only robot that can act as leader. The robots on the leaves move up, then r_l moves toward v.

during the emptying of the subtree. In particular, in Fig. 15.(a), r_l moves inside the first subtree making it different from all the other subtrees. In Fig. 15.(b), robots $r_2, \ldots, r_n$ and the ones on the leaves cautiously move one step toward v making the subtrees different from any configuration that might be created by the first subtree in the successive movements. In Fig. 15.(c), r_l and r_1 move out of the subtree. Then all the robots that previously moved toward the root make a step back to their starting position. From here, the strategy can be repeated to move all the robots in the overloaded subtrees toward the leaves.

However a variation of the configuration of Fig. 15.(a), shown in Fig. 17.(a) is unsolvable in SSync. Even though a leader robot is present, it is not possible to move the robots in the other two subtrees so as to generate a configuration different from each configuration generated in the first subtree during its emptying. In fact, the robots on the leaves cannot move. Other unsolvable configurations can be generated when the leader robot cannot move, as shown in Fig. 17.(c).

In any case, in order to solve GMV from a configuration C with a critical vertex v, it is necessary to solve the sub-problem EmptyRoots defined as follows: if v is critical, $T(v, v_1), T(v, v_2), \ldots, T(v, v_k)$ are pairwise isomorphic critical-subtrees of v, and $v_1, v_2, \ldots, v_k$ are occupied, then each resolving algorithm for GMV must be able to transform C into a configuration C' in which $v_1, v_2, \ldots, v_k$ are all unoccupied. This is necessary so that a leader robot can move onto one of them in order to break the symmetry among the critical-subtrees. Notice that the only way to solve EmptyRoots is moving robots located in v_i downward to the corresponding critical-subtrees $T(v, v_i)$, $i = 1, \ldots, k$.

Sometimes solving EmptyRoots implies the movement of other robots (see Fig. 16.(a)), in other cases EmptyRoots cannot be solved (see Fig. 16.(b)). In Fig. 16.(a), if the robot on v_{r_2} moves downward as first move, the configuration becomes unsolvable; in fact, the vertex left by r_2 becomes a critical vertex being the two subtrees (below such a vertex) isomorphic and overloaded. Moreover, EmptyRoots must be solved for these two critical-subtrees as well, as this cannot be done without incurring in collisions. On the other hand, the configuration of Fig. 16.(a), can be solved by first moving robot r_1 upward and then moving r_2 downward. The movement of r_1 makes the left subtree of v_{r_2} different from the one on its right. On the contrary, the configuration shown in Fig. 16.(b) is unsolvable because it is not possible to design a preliminary move in order to differentiate the subtrees of v'_{r_2} before moving r_2 downward. We conjecture that deciding whether the sub-problem EmptyRoots can be solved could require exploring a very large number of possible moves (even an exponential number).

3. For GMV, FSync robots are more powerful. Consider the configuration of Fig. 17.(a). Since r_2 and r_3 are pairwise equivalent robots, it is not reasonable to let them both move concurrently. Instead, as shown in Fig. 17.(b), r_l and r_1 can move concurrently toward v (we recall the reader that here we assume FSync robots). From there, r_l can move toward a leaf whereas r_1 can start playing the role previously played by r_l, i.e., it can move downward toward another subtree and grab another robot outside to make its role. In general, if there were $n-1$ critical-subtrees, by repeating this strategy, all the robots can be correctly moved to the leaves. Notice that this strategy cannot be implemented in SSync since from the configuration shown in Fig. 17.(b) the adversary could move only r_l back to v creating a loop in the algorithm or only r_1, creating a multiplicity with r_l.

As an additional example, consider the configuration shown in Fig. 17(c). It represents a case in which there is a critical vertex v because of exactly two critical-subtrees and it is possible to elect a leader. We have already discussed a similar case in which the leader robot can move within one critical-subtree to break the symmetry and hence to solve GMV. We now show that here the problem cannot be solved in SSync but it is solvable in FSync. In fact, in SSync, moving r_l or the two equivalent robots r_1 and r_2 would make the three subtrees all isomorphic. Hence, in any case, v remains critical and the obtained configuration does not contain any more leader robots. Instead, in FSync, the simultaneous movement of r_1, r_2 and r_l, as shown in the figure, allows to solve the problem. In fact, while r_l moves toward v, both r_1 and r_2 can move downward, and the achieved configuration becomes similar to that in Fig. 14, where the leader robot can enter a critical-subtree to break the symmetry. Clearly, the combined movement described cannot be applied in SSync.

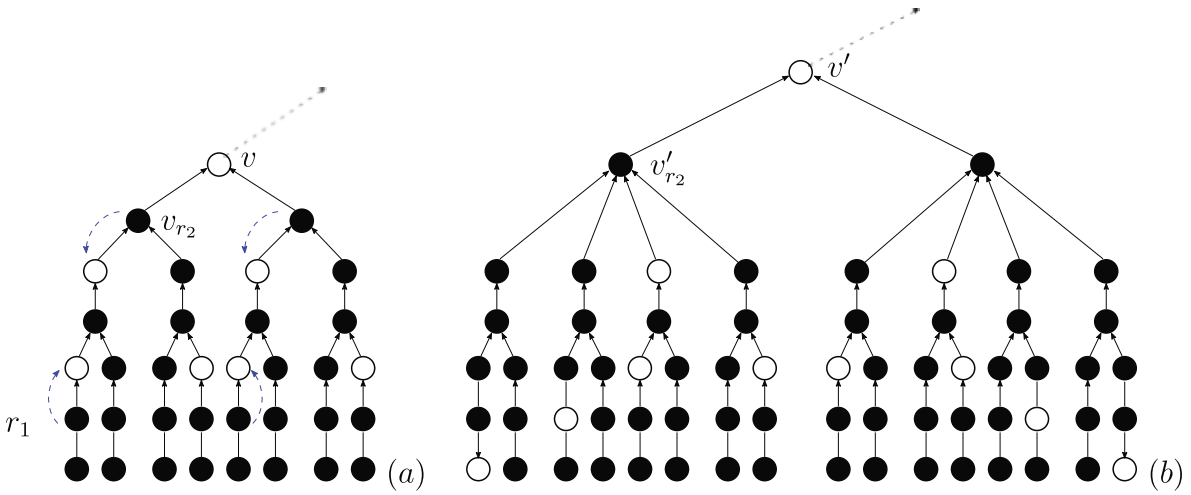


Fig. 16. Both figures show configurations in which GMV requires solving the sub-problem EMPTYROOTS. (*a*) A solvable configuration where the preliminary move of r_1 upward followed by the move of r_2 downward maintains the difference between the subtrees currently rooted in v_{r_2}. (*b*) An unsolvable configuration.

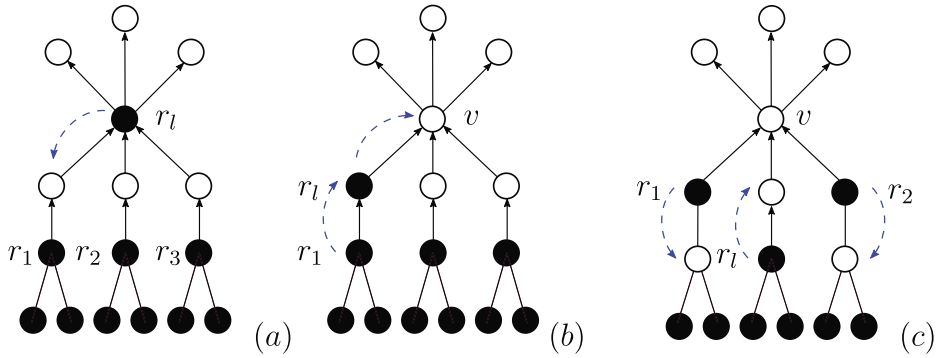


Fig. 17. Three figures referring to symmetric configurations solvable in FSYNC but not in SSYNC.

6. Conclusion

We have introduced the GEODESIC MUTUAL VISIBILITY problem in the context of robots moving along the edges of a graph, and in particular, on grids and trees, operating under the LCM model. Regarding capabilities, robots are rather weak, as they are oblivious and without any direct means of communication.

For (finite or infinite) grids, robots are considered to be synchronous and endowed with chirality. Under these conditions, we have shown that GMV_{area} can be solved by a time-optimal distributed algorithm.

For trees, robots are considered to be semi-synchronous and we have shown that GMV can be solved when the initial configuration does not admit critical vertices. Concerning the time complexity, the algorithm leaves some gaps with the proposed lower bound.

This work opens a wide research area concerning GMV on other graph topologies or even on general graphs. However, difficulties may arise in moving robots in the presence of symmetries. Then, the study of GMV in asymmetric graphs or graphs with a limited number of symmetries deserves main attention, as well as the removal of the assumption about a common chirality. Other directions concern deeper investigations into the different types of schedulers: synchronous, semi-synchronous or asynchronous. Furthermore, for graphs with an embedding in a Euclidean space, the introduction of obstructed visibility during the Look phase might be of main interest, i.e., the property that if three robots are collinear, the one in the middle obstruct the reciprocal visibility of the other two.

Declaration of competing interest

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

Data availability

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

References

[1] S. Cicerone, A. Di Fonso, G. Di Stefano, A. Navarra, The geodesic mutual visibility problem for oblivious robots: the case of trees, in: Proc. 24th International Conference on Distributed Computing and Networking (ICDCN), ACM, 2023, pp. 150–159, http://dx.doi.org/10.1145/3571306.3571401.

[2] S. Cicerone, A. Di Fonso, G. Di Stefano, A. Navarra, Time-optimal geodesic mutual visibility of robots on grids within minimum area, in: Proc. 25th International Symposium on Stabilization, Safety, and Security of Distributed System (SSS), in: LNCS, 2023, in press.

[3] S. Cicerone, G. Di Stefano, A. Navarra, A structured methodology for designing distributed algorithms for mobile entities, Inform. Sci. 574 (2021) 111–132, http://dx.doi.org/10.1016/j.ins.2021.05.043.

[4] P. Flocchini, G. Prencipe, N. Santoro (Eds.), Distributed Computing By Mobile Entities, Current Research in Moving and Computing, in: Lecture Notes in Computer Science, vol. 11340, Springer, 2019, http://dx.doi.org/10.1007/978-3-030-11072-7.

[5] G. D'Angelo, G. Di Stefano, R. Klasing, A. Navarra, Gathering of robots on anonymous grids and trees without multiplicity detection, Theoret. Comput. Sci. 610 (2016) 158–168.

[6] S. Cicerone, A. Di Fonso, G. Di Stefano, A. Navarra, Arbitrary pattern formation on infinite regular tessellation graphs, Theoret. Comput. Sci. 942 (2023) 1–20, http://dx.doi.org/10.1016/j.tcs.2022.11.021.

[7] S. Cicerone, G. Di Stefano, A. Navarra, Asynchronous arbitrary pattern formation: the effects of a rigorous approach, Distrib. Comput. 32 (2) (2019) 91–132.

[8] S. Cicerone, G. Di Stefano, A. Navarra, Embedded pattern formation by asynchronous robots without chirality, Distrib. Comput. 32 (4) (2019) 291–315.

[9] S. Cicerone, G. Di Stefano, A. Navarra, Gathering robots in graphs: The central role of synchronicity, Theoret. Comput. Sci. 849 (2021) 99–120, http://dx.doi.org/10.1016/j.tcs.2020.10.011.

[10] S. Das, P. Flocchini, N. Santoro, M. Yamashita, Forming sequences of geometric patterns with oblivious mobile robots, Distrib. Comput. 28 (2) (2015) 131–145.

[11] P. Flocchini, G. Prencipe, N. Santoro, G. Viglietta, Distributed computing by mobile robots: uniform circle formation, Distrib. Comput. 30 (2017) 413–457.

[12] M. Yamashita, I. Suzuki, Characterizing geometric patterns formable by oblivious anonymous mobile robots, Theoret. Comput. Sci. 411 (26–28) (2010) 2433–2453.

[13] S. Bhagat, Optimum algorithm for the mutual visibility problem, in: WALCOM: Algorithms and Computation - 14th International Conference, WALCOM 2020, in: Lecture Notes in Computer Science, vol. 12049, Springer, 2020, pp. 31–42, http://dx.doi.org/10.1007/978-3-030-39881-1_4.

[14] S. Bhagat, K. Mukhopadhyaya, Mutual visibility by robots with persistent memory, in: Proc. 13th Int.'L Workshop on Frontiers in Algorithmics (FAW), in: Lecture Notes in Computer Science, vol. 11458, Springer, 2019, pp. 144–155.

[15] G.A. Di Luna, P. Flocchini, S.G. Chaudhuri, F. Poloni, N. Santoro, G. Viglietta, Mutual visibility by luminous robots without collisions, Inform. and Comput. 254 (2017) 392–418.

[16] G.A. Di Luna, P. Flocchini, F. Poloni, N. Santoro, G. Viglietta, The mutual visibility problem for oblivious robots, in: Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014, Carleton University, Ottawa, Canada, 2014.

[17] G. Sharma, C. Busch, S. Mukhopadhyay, Mutual visibility with an optimal number of colors, in: Proc. 11th Int.'l Symp. on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS), in: Lecture Notes in Computer Science, vol. 9536, Springer, 2015, pp. 196–210.

[18] R. Adhikary, K. Bose, M.K. Kundu, B. Sau, Mutual visibility by asynchronous robots on infinite grid, in: Algorithms for Sensor Systems - 14th Int. Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2018, in: Lecture Notes in Computer Science, vol. 11410, Springer, 2018, pp. 83–101, http://dx.doi.org/10.1007/978-3-030-14094-6_6.

[19] S. Bhagat, S.G. Chaudhuri, K. Mukhopadhyaya, Mutual visibility for asynchronous robots, in: Proc. 26th Int.'L Colloquium on Structural Information and Communication Complexity (SIROCCO), in: Lecture Notes in Computer Science, vol. 11639, Springer, 2019, pp. 336–339.

[20] S. Bhagat, K. Mukhopadhyaya, Optimum algorithm for mutual visibility among asynchronous robots with lights, in: Proc. 19th Int.'l Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS), in: Lecture Notes in Computer Science, vol. 10616, Springer, 2017, pp. 341–355.

[21] P. Poudel, A. Aljohani, G. Sharma, Fault-tolerant complete visibility for asynchronous robots with lights under one-axis agreement, Theoret. Comput. Sci. 850 (2021) 116–134, http://dx.doi.org/10.1016/j.tcs.2020.10.033.

[22] G. Sharma, R. Vaidyanathan, J.L. Trahan, Optimal randomized complete visibility on a grid for asynchronous robots with lights, Int. J. Netw. Comput. 11 (1) (2021) 50–77.

[23] P. Poudel, G. Sharma, A. Aljohani, Sublinear-time mutual visibility for fat oblivious robots, in: Proceedings of the 20th International Conference on Distributed Computing and Networking, ICDCN 2019, ACM, 2019, pp. 238–247, http://dx.doi.org/10.1145/3288599.3288602.

[24] A. Aljohani, P. Poudel, G. Sharma, Complete visitability for autonomous robots on graphs, in: 2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018, IEEE Computer Society, 2018, pp. 733–742, http://dx.doi.org/10.1109/IPDPS.2018.00083.

[25] G. Di Stefano, Mutual visibility in graphs, Appl. Math. Comput. 419 (2022) 126850, http://dx.doi.org/10.1016/j.amc.2021.126850.

[26] S. Cicerone, G. Di Stefano, S. Klavzar, On the mutual visibility in cartesian products and triangle-free graphs, Appl. Math. Comput. 438 (2023) 127619, http://dx.doi.org/10.1016/j.amc.2022.127619.

[27] S. Cicerone, G. Di Stefano, L. Drožek, J. Hedžet, S. Klavžar, I.G. Yero, Variety of mutual-visibility problems in graphs, Theoret. Comput. Sci. 974 (2023) 114096, http://dx.doi.org/10.1016/j.tcs.2023.114096.

[28] P. Flocchini, G. Prencipe, N. Santoro (Eds.), Distributed Computing by Mobile Entities, Current Research in Moving and Computing, in: LNCS, vol. 11340, Springer, 2019, http://dx.doi.org/10.1007/978-3-030-11072-7.

[29] S. Cicerone, G. Di Stefano, A. Navarra, "Semi-asynchronous": A new scheduler in distributed computing, IEEE Access 9 (2021) 41540–41557, http://dx.doi.org/10.1109/ACCESS.2021.3064880.

[30] M. D'Emidio, G. Di Stefano, D. Frigioni, A. Navarra, Characterizing the computational power of mobile robots on graphs and implications for the euclidean plane, Inform. and Comput. 263 (2018) 57–74.

[31] S. Cicerone, G. Di Stefano, A. Navarra, Solving the pattern formation by mobile robots with chirality, IEEE Access 9 (2021) 88177–88204, http://dx.doi.org/10.1109/ACCESS.2021.3089081.

[32] I. Suzuki, M. Yamashita, Distributed anonymous mobile robots: Formation of geometric patterns, SIAM J. Comput. 28 (4) (1999) 1347–1363.

[33] N. Santoro, Design and Analysis of Distributed Algorithms, John Wiley & Sons, 2007.

[34] S.R. Buss, Alogtime algorithms for tree isomorphism, comparison, and canonization, in: Proc. 5th Kurt Gödel Colloquium on Computational Logic and Proof Theory (KGC), in: Lecture Notes in Computer Science, vol. 1289, Springer, 1997, pp. 18–33.

[35] G.L. Miller, J.H. Reif, Parallel tree contraction, part 2: Further applications, SIAM J. Comput. 20 (6) (1991) 1128–1147, http://dx.doi.org/10.1137/0220070.