The Geodesic Mutual Visibility Problem for Oblivious Robots: the case of Trees

Serafino Cicerone
serafino.cicerone@univaq.it
University of L'Aquila
L'Aquila, Italy

Alessia Di Fonso
alessia.difonso@graduate.univaq.it
University of L'Aquila
L'Aquila, Italy

Gabriele Di Stefano
gabriele.distefano@univaq.it
University of L'Aquila
L'Aquila, Italy

Alfredo Navarra
alfredo.navarra@unipg.it
University of Perugia
Perugia, Italy

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. Here we introduce the Geodesic Mutual Visibility problem, a possible counterpart for robots moving in a discrete environment. For a set of robots disposed on the vertices of a graph, it asks for moving the robots so that they are pairwise geodesic mutually visible, that is there is a shortest path (i.e., a "geodesic") between each pair of robots along which no other robots reside.

It is known that, for tree topologies, the maximum number of robots that can be disposed by respecting the geodesic mutual visibility equals the number of leaves. Given a tree T with $\ell(T)$ leaves, we consider $n \leq \ell(T)$ identical, mobile and semi-synchronous robots disposed on n different vertices of T, and we study the problem to make the robots move so as to reach a configuration where the geodesic mutual visibility is achieved.

We propose a distributed algorithm, along with its correctness, for configurations not admitting *critical* vertices. Intuitively, a vertex v is critical if two or more equivalent robots must pass through v in order to reach a leaf, hence potentially collide in v. Furthermore, we provide an extensive discussion about the implications as well as complications arising when admitting critical vertices and/or the synchronous or the asynchronous settings, along with impossibility results or possible strategies of resolution to investigate.

CCS CONCEPTS

• **Theory of computation** → **Distributed algorithms**; • **Mathematics of computing** → *Trees*; • **Networks** → Network structure.

ICDCN 2023, January 4–7, 2023, Kharagpur, India

ACM ISBN 978-1-4503-9796-4/23/01…$15.00
https://doi.org/10.1145/3571306.3571401

KEYWORDS

Autonomous mobile robots, Oblivious robots, Mutual visibility, Trees

ACM Reference Format:
Serafino Cicerone, Alessia Di Fonso, Gabriele Di Stefano, and Alfredo Navarra. 2023. The Geodesic Mutual Visibility Problem for Oblivious Robots: the case of Trees. In *24th International Conference on Distributed Computing and Networking (ICDCN 2023), January 4–7, 2023, Kharagpur, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3571306.3571401

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., [20]). 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., [8–11, 13, 18, 19, 27].

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 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 the 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 [3], i.e., they are always all active and perform each computational cycle within a same amount of time; semi-synchronous [6, 15, 16, 25], 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 [1, 4, 5, 15, 22, 25, 26], i.e., each robot can be activated at any time and the duration of its computational

cycle is finite but unkown. 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 [1, 26] 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 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 collision. Concerning the last motivation, in [2] 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 grids and infinite hexagonal grids, both embedded in the Euclidean plane.

Recently, the geodesic mutual visibility has been investigated in [17] 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 $\mu(G)$. For instance, within a path P only two robots can be placed, i.e., $\mu(P) = 2$, whereas for a ring R, $\mu(R) = 3$. In a general graph G, it turns out to be NP-complete to compute $\mu(G)$, however for a tree T, it has been proven that $\mu(T) = \ell(T)$, with $\ell(T)$ being the number of leaves of T.

After formally defining the problem of achieving GMV starting from a configuration of robots disposed on general graphs, we focus on tree topologies. 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 robots without any explicit means of communication nor memory of past events, i.e., they are oblivious and without lights. Hence, the movement of a robot does rely only on local computations on the basis of the acquired snapshot during the `Look` phase. 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 equivalent 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 general case with $n \leq \ell(T)$.

Furthermore, we provide an extended discussion about the configurations admitting critical vertices as well as for 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.

1.1 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, introducing also useful notation such as the formal definition of critical vertex; Section 4 describes the proposed resolution algorithm along with the correctness proof; Section 4.6 discusses on the complexity of the designed algorithm by also providing a lower bound for GMV; Section 5 provides challenging scenarios to highlight difficulties arising when critical vertices are admitted; finally, Section 6 concludes the paper, posing interesting future work directions.

2 ROBOT MODEL

We assume the standard $\mathcal{OBLOT}$ model [20]. In particular, robots are considered to be *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* algorithm), *silent* (no means of direct communication), and *disoriented* (no common coordinate system, no chirality).

The ability to solve a given task depends also on the magnitude of synchrony among the robots. One of the most studied scenario assumes robots alternate within two main states: *active* or *inactive*. When active, a robot executes a `Look-Compute-Move` (`LCM`) cycle by performing the following three operations in sequence, each of them associated with a different state:

- `Look`: The robot r observes the environment. The result of this phase is a snapshot of the positions of all robots with respect to its own perception.
- `Compute`: The robot r executes the designed algorithm, using the data sensed in the `Look` phase as input. When robots are on graphs, the result of this phase either is the vertex v_r where r currently resides or it is a vertex among those at one hop distance (i.e., at most one edge can be traversed).
- `Move`: The robot r moves toward the computed target. If the target is v_r, then the robot stays still, i.e., it performs what is called *nil* movement.

Moves on graphs are always considered as *instantaneous*. This results in always perceiving robots on vertices and never on edges

during Look phases. The rationale behind this assumption is that the graph may model a communication network, whereas robots model software agents.

In the literature, different characterizations of the environment have been considered according whether robots are fully-synchronous, semi-synchronous, or asynchronous (cf. [12, 20]). 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 cycles 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 to ever move.

It is worth to remark that the three synchronization schedulers induce the following hierarchy (see [14]): FSync robots are more powerful (i.e., they can solve more tasks) than SSync robots, that in turn are more powerful that 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 determine a sequence of specific time instants $t_0 < t_1 < t_2 < \ldots$ 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 an algorithm $\mathcal{A}$ 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, given an algorithm $\mathcal{A}$, in SSync or Async there exists more than one execution 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

The topology where robots are placed on 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.

As described in Introduction, we study GMV in the context of trees. In the rest of this section we provide the necessary concepts.

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 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 that the adversary activates two equivalent robots at the same time and that they perform the same move simultaneously.

3.2 GMV on trees

Given a configuration $C = (T, \lambda)$, a vertex v of T such that $\lambda(v) > 0$ is called *occupied*, *unoccupied* otherwise. If $\lambda(v) \geq 2$, there is a *multiplicity* in v. Given a vertex v, 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, $i = 1, 2, \ldots, k$. If $e = (v_1, v_2)$ is an edge of T, the

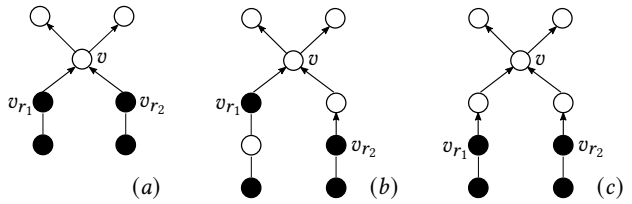

Figure 1: Examples of configurations. Occupied vertices are represented in black.

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 3.1. 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'.

Figure 1 shows three configurations where the complete subtrees of vertex v with robots are overloaded subtrees.

Definition 3.2. 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.

Consider again Figure 1. 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 3.3. 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 Figure 1.(a) is critical (in fact, $(T(v, v_{r_1}), \lambda)$ and $(T(v, v_{r_2}), \lambda)$ are isomorphic and both overloaded). In Figure 1.(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, could generate unsolvable configurations).

Definition 3.4. Let $C = (T, \lambda)$ be a configuration, where $T = (V, E)$ is a tree. C is *initial* if it contains $n \leq \ell(T)$ robots, each one lying on a different vertex – that is, $\lambda(v) \leq 1$ for each $v \in V$ – and C does not contain critical vertices. C is *final* if it contains n robots each one lying on a distinct leaf of T.

According to this definition, the specific version of GMV addressed in this work can be defined as the problem of transforming an initial configuration into a final one. Notice that a final configuration has always robots positioned on the leaves, and this ensures that GMV is solved even if the problem definition does not require such a property. In particular, given an initial configuration C, a deterministic distributed algorithm $\mathcal{A}$ *resolves* GMV if for any execution $\mathbb{E} : C = C(t_0), C(t_1), \ldots$ of $\mathcal{A}$, there exists a time instant $t^* \geq 0$ such that $C(t^*)$ is final and no robots move after t^*, i.e., $C(t) = C(t^*)$ holds for all $t \geq t^*$. Given a configuration $C = (G, \lambda)$, we say that GMV *is solvable from* C if and only if there exists a resolving algorithm for C. We will prove the following result:

Theorem 3.5. *Let $C = (T, \lambda)$ be an initial configuration composed of $n \leq \ell(T)$ SSync robots, then* GMV *is solvable from C.*

Actually, we will provide the formal proof of the above theorem by considering the extreme case with $n = \ell(T)$ robots (cf., Theorem 4.6). Successively, we give the necessary modifications for the general case with $n \leq \ell(T)$.

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 in this first work in which we deal with GMV, we consider symmetric configurations, but only those without critical vertices. In Section 5, we provide an extensive discussion in which we motivate how the presence of critical vertices makes solving GMV particularly difficult, if not even impossible.

4 A RESOLVING ALGORITHM FOR GMV

In this section, we provide the proposed resolution algorithm along with its correctness proof, that is a proof for Theorem 3.5.

4.1 Further notation and definitions

In the following, given an initial configuration $C = (T, \lambda)$, we denote by $R = \{r_1, r_2, \ldots, r_{\ell(T)}\}$ the set of robots in C.[1]

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 [24]. The provided algorithm requires that each robot identifies a single vertex as center, denoted as $c(T)$. 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.

In the following, given an initial configuration C, 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 initial configuration C where GMV is not yet solved.

[1]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, hence no algorithm can take advantage of names of elements in R.

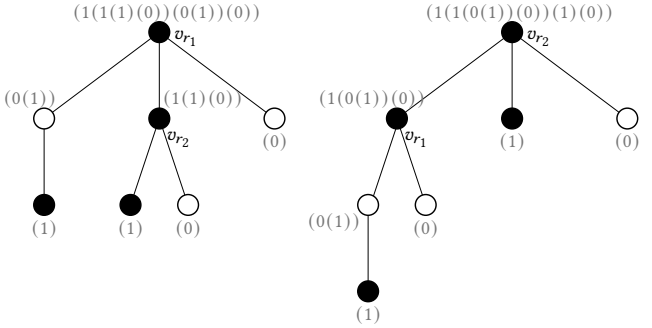

Figure 2: Examples of views associated with different robots/vertices.

Definition 4.1. Let $C = (T, \lambda)$ be an initial configuration. $R'(C)$ is the set containing any robot $r \in R$ such that:

- 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 GMV 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, $s(P) = \sum_{e \in P} s(e)$.

4.2 View of a robot

In the algorithm, sometimes we need to distinguish among robots having some properties (e.g., minimum distance from unoccupied leaves). To this purpose, we use the so-called *view* of a robot. The view of each robot is elaborated during the `Compute` phase, starting from the snapshot perceived during the `Look` phase. In particular, we consider an approach similar to that used in [7, 21] 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 Figure 2 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.

4.3 Description of the algorithm

The provided algorithm for solving GMV from each initial configuration C is called `MoveToLeaf` and is described in Algorithm 1. Essentially, we can assume that, during the `LCM`-cycle, each robot first computes 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`.

Algorithm 1 `MoveToLeaf`

Input: Initial configuration $C = (T, \lambda)$.
1: compute the view of C
2: compute $R'(C)$
3: **if** $R'(C) \neq \emptyset$ **then**
4: *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
5: **else**
6: let $\mathcal{S}$ = `DetermineMovingRobots`(C)
7: **if** $\mathcal{S} \neq \emptyset$ **then**
8: *move* m_2*:* let (v, v_r) be the key in $\mathcal{S}$, with r of minimum view, r moves toward v

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 $c(T)$ towards unoccupied leaves which can be brought to target by moving them along oriented paths. This case can be observed in Figure 3, where robots r_1 and r_2 belong to $R'(C)$. In this situation, the algorithm preliminarily moves these robots (cf. move m_1 at Line 4) until a configuration C_1 in which $R'(C_1) = \emptyset$ is generated.

Algorithm 2 `DetermineMovingRobots`

Input: Initial configuration $C = (T, \lambda)$.
1: let $\mathcal{S}$ be an empty map
2: compute $\mathcal{P}(C)$
3: **for all** $P \in \mathcal{P}(C)$ **do**
4: let l be the leaf where P leads
5: 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
6: **for all** occupied vertex $v_r \in T(v, u)$ **do**
7: **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**
8: let $\mathcal{S}[(v, v_r)] = (s((v_0, v_1)), s((v_1, v_2)), \ldots, s((v_{t-1}, v_r))$
9: let $\mathcal{S}'$ be the submap of $\mathcal{S}$ containing the lexicographically minimal sequences of $\mathcal{S}$
10: **return** $\mathcal{S}'$

Successively, since R' is empty, `MoveToLeaf` calls procedure `DetermineMovingRobots` at Line 6, the routine described in Algorithm 2. 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 Figure 3). In this way, a sequence of integers is 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 to perform "bad moves" that could transform vertices from potentially-critical to critical thus generating unsolvable configurations. For instance, the rightmost sequence represented in Figure 3, 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. 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.

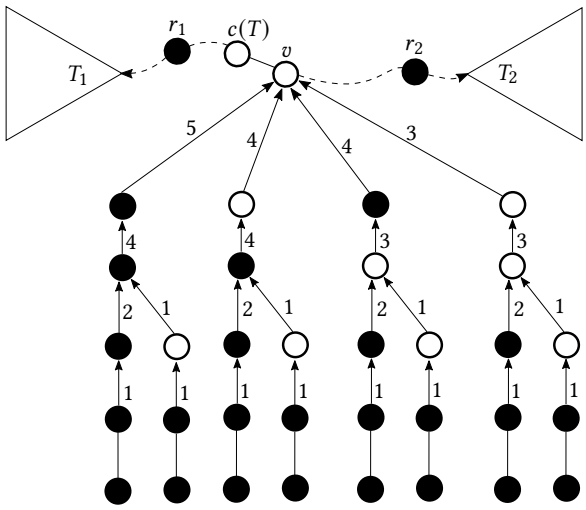

Figure 3: A schematic representation of an initial configuration C elaborated by `MoveToLeaf`. The triangles represent two subtrees denoted as T_1 and T_2 and containing unoccupied leaves. The dashed and curved lines represent paths. In the discussion it is assumed that the paths from $c(T)$ to T_1 and from $c(T)$ to T_2 belong to $\mathcal{P}(C)$.

4.4 Correctness

In this section, we prove that `MoveToLeaf` is a resolving algorithm for GMV from each initial configuration C. This is achieved by exploiting a series of lemmata concerning useful properties.

Lemma 4.2. *$\mathcal{P}(C) \neq \emptyset$ for each initial and non-final configuration $C = (T, \lambda)$.*

Proof. Since the configuration is non-final, 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 4.3. *Let $C = (T, \lambda)$ be an initial configuration, and let C' be the configuration generated from C by `MoveToLeaf` according to one execution of move m_1. Then, C' is still an initial configuration.*

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'', is initial, i.e., it does not contain critical vertices nor multiplicities. By Lemma 4.2 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 4.4. *Let $C = (T, \lambda)$ be an initial configuration in which $R'(C) = \emptyset$. 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 exists 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 4.1. Hence $R'(C) \neq \emptyset$, against the assumption. □

Lemma 4.5. *Let $C = (T, \lambda)$ be an initial configuration, and let C' be the configuration generated from C by `MoveToLeaf` according to one execution of move m_2. Then, C' is still an initial configuration.*

Proof. Since `MoveToLeaf` applies move m_2 then $R'(C) = \emptyset$. By Lemma 4.4, 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'', is initial, i.e., it does not contain 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 4.4, 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 since C is an initial configuration. □

Theorem 4.6. Let $C = (T, \lambda)$ be an initial configuration composed of $n = \ell(T)$ SSync robots, then GMV is solvable from C.

Proof. We prove that `MoveToLeaf` is a resolving algorithm for C. 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 wlog $R'(C(0)) \neq \emptyset$. This implies that `MoveToLeaf` applies m_1 to $C(0)$. The resulting configuration $C(1)$ is still initial (cf. Lemma 4.3) and $s(C(1)) < s(C(0))$ because m_1 moves robots toward unoccupied leaves along oriented edges. Hence, by repeatedly applying m_1, `MoveToLeaf` leads to an initial configuration $C(t')$, $t' > 0$, in which $R'(C(t')) = \emptyset$. In $C(t')$, `MoveToLeaf` applies move m_2 and Lemma 4.5 guarantees that $C(t' + 1)$ is still initial. 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 4.4 we get $R'(C(t'')) \neq \emptyset$ and move m_1 is applied again.

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 there exists a time $t^* > 0$ such that $s(C(t^*)) = 0, R(C(t^*)) = \emptyset$, and the map $\mathcal{S}$ is empty. Hence, $C(t^*)$ is final and no further moves are made. This means that `MoveToLeaf` is able to solve GMV from C. □

4.5 General 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 change, so as their placement.

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 has been solved.

By Theorem 4.6 and the above discussion, we conclude the general Theorem 3.5 holds.

4.6 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 4.7. `MoveToLeaf` *requires* $O(nD)$ *epochs to solve* GMV *from initial configurations composed of n* SSync *robots.*

Proof. The statement simply follows by observing that procedure `MoveToLeaf`, at Line 4 or at Line 8, 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 4.8. GMV *requires* $\Omega(n + D)$ *epochs to be solved from initial configurations composed of n* SSync *robots.*

Proof. Let $C = (T, \lambda)$ be an initial configuration 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 $m >> n$, 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 fist 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 is solved. This implies that any solving algorithm requires $\Omega(n + D)$ epochs to solve GMV on the assumed initial configuration. □

5 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 Figure 1.(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 5.1. Let C be an initial configuration in which 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 move within) any of the critical-subtrees then a multiplicity is created.

Figure 1.(c) shows another unsolvable configuration in which the previous claim does not applies. Theoretically, if other robots

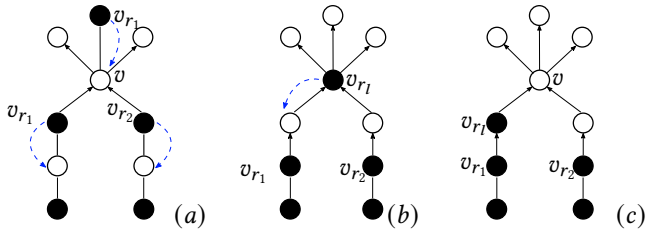

Figure 4: (a): A configuration with a critical vertex v. The dashed arrows show the direction proposed for the movement of the robots in order to solve GMV**; (b): the symmetric configuration obtained after the one on the left with a new proposed movement; (c): the configuration made asymmetric.**

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 Figure 1.(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 Figure 4.(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 Figure 1.(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 Figure 4.(b), r_l can move toward one of the two symmetric subtrees, hence breaking the symmetry, and thus obtaining the configuration in Figure 4.(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 5.2. Let C be an initial configuration in which there is a critical vertex but no leader robots. Then, GMV cannot be solved from C even by FSync robots.

Figure 5.(a) shows a more general case with respect to Figure 4 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 Figure 5.(a) and then the algorithm creates a configuration

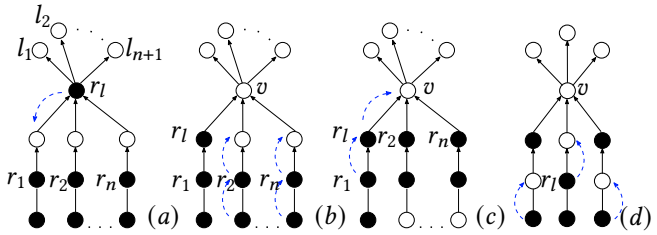

Figure 5: (a): A symmetric configurations with more than two isomorphic overloaded subtrees. Vertex v is critical, r_l is a leader robot, r_l moves inside the first subtree (b): robots of other subtrees, move toward v, (c): the first subtree can be emptied. (d): v is critical and r_l is the only robot that can act as leader. The robots on the leaves move up, then r_l moves toward v.

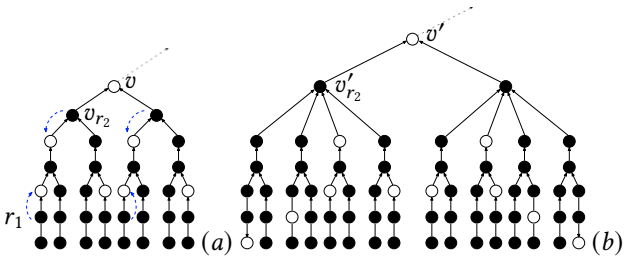

Figure 6: Both figures show configurations in which GMV requires solving the sub-problem EmptyRoots. (a) A solvable configuration where the preliminary move of r_1 upward followed by the move of r_2 downward maintains the difference between the subtrees currently rooted in v_{r_2}. (b) An unsolvable configuration.

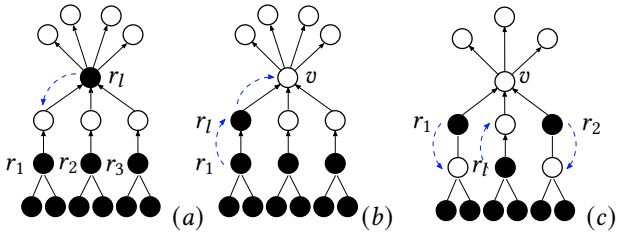

Figure 7: Three figures referring to symmetric configurations solvable in FSync but not in SSync.

on other critical-subtrees so that to be different from any other configuration created on the first subtree during the emptying of the subtree. In particular in Figure 5.(a), r_l moves inside the first subtree making it different from all the other subtrees. In Figure 5.(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 Figure 5.(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 Figure 5.(a), shown in Figure 7.(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 Figure 7.(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 in to 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 Figure 6.(a)), in other cases EmptyRoots cannot be solved (see Figure 6.(b)). In Figure 6.(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 the Figure 6.(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 Figure 6.(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 Figure 7.(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 Figure 7.(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 Figure 7.(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 Figure 7.(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, by 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 anymore 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 Figure 4, where the leader robot can enter in a critical-subtree to break the symmetry. Clearly, the combined movement described cannot be applied 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 trees, operating under the LCM model. About capabilities, robots are rather weak, as they are SSync and oblivious. The only restriction imposed to the initial configuration is the absence of critical vertices.

We have proposed a deterministic and distributed algorithm to solve GMV on trees. Furthermore, we have provided an extensive discussion about challenging directions for future research. As a first open question, it remains to show a full characterization about GMV within SSync. To this respect, the main difficulty arising within any of the available schedulers certainly comes from managing critical vertices. We have provided some hints about possible approaches, but it seems rather difficult to find out a general strategy to resolve such occurrences. As we have seen, this is especially exacerbated within FSync.

It is also interesting to investigate on the time complexity of the resolution algorithms. In fact, we have shown there is a big gap between the lower bound of $\Omega(n + D)$ and the complexity of the proposed algorithm upper bounded by $O(nD)$.

As a wider research area, one could investigate GMV on other topologies or even on general graphs. However, as already pointed out, difficulties arise in moving robots in presence of symmetries. Then the study of GMV in asymmetric graphs or graphs with a limited number of symmetries deserves main attention, even in the Async case.

ACKNOWLEDGMENTS

The work has been supported in part by the Italian National Group for Scientific Computation (GNCS-INdAM).

REFERENCES

[1] Ranendu Adhikary, Kaustav Bose, Manash Kumar Kundu, and Buddhadeb Sau. 2018. 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 (Lecture Notes in Computer Science, Vol. 11410)*. Springer, 83–101.

[2] Aisha Aljohani, Pavan Poudel, and Gokarna Sharma. 2018. 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, 733–742.

[3] Subhash Bhagat. 2020. Optimum Algorithm for the Mutual Visibility Problem. In *WALCOM: Algorithms and Computation - 14th International Conference, WALCOM 2020 (Lecture Notes in Computer Science, Vol. 12049)*. Springer, 31–42.

[4] Subhash Bhagat, Sruti Gan Chaudhuri, and Krishnendu Mukhopadhyaya. 2019. Mutual Visibility for Asynchronous Robots. In *Proc. 26th Int.'l Colloquium on Structural Information and Communication Complexity (SIROCCO) (Lecture Notes in Computer Science, Vol. 11639)*. Springer, 336–339.

[5] Subhash Bhagat and Krishnendu Mukhopadhyaya. 2017. 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) (Lecture Notes in Computer Science, Vol. 10616)*. Springer, 341–355.

[6] Subhash Bhagat and Krishnendu Mukhopadhyaya. 2019. Mutual Visibility by Robots with Persistent Memory. In *Proc. 13th Int.'l Workshop on Frontiers in Algorithmics (FAW) (Lecture Notes in Computer Science, Vol. 11458)*. Springer, 144–155.

[7] Samuel R. Buss. 1997. Alogtime Algorithms for Tree Isomorphism, Comparison, and Canonization. In *Proc. 5th Kurt Gödel Colloquium on Computational Logic and Proof Theory (KGC) (Lecture Notes in Computer Science, Vol. 1289)*. Springer, 18–33.

[8] Serafino Cicerone, Alessia Di Fonso, Gabriele Di Stefano, and Alfredo Navarra. 2021. Arbitrary Pattern Formation on Infinite Regular Tessellation Graphs. In *Proc. 22nd Int.'l Conf. on Distributed Computing and Networking (ICDCN)*. ACM, New York, NY, USA, 56–65.

[9] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. 2019. Asynchronous Arbitrary Pattern Formation: the effects of a rigorous approach. *Distributed Computing* 32, 2 (2019), 91–132.

[10] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. 2019. Embedded pattern formation by asynchronous robots without chirality. *Distributed Computing* 32, 4 (2019), 291–315.

[11] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. 2021. Gathering robots in graphs: The central role of synchronicity. *Theor. Comput. Sci.* 849 (2021), 99–120.

[12] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. 2021. "Semi-Asynchronous": A New Scheduler in Distributed Computing. *IEEE Access* 9 (2021), 41540–41557.

[13] Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. 2015. Forming sequences of geometric patterns with oblivious mobile robots. *Distributed Computing* 28, 2 (2015), 131–145.

[14] Mattia D'Emidio, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. 2018. Characterizing the computational power of mobile robots on graphs and implications for the Euclidean plane. *Inf. Comput.* 263 (2018), 57–74.

[15] Giuseppe Antonio Di Luna, Paola Flocchini, Sruti Gan Chaudhuri, Federico Poloni, Nicola Santoro, and Giovanni Viglietta. 2017. Mutual visibility by luminous robots without collisions. *Inf. Comput.* 254 (2017), 392–418.

[16] Giuseppe Antonio Di Luna, Paola Flocchini, Federico Poloni, Nicola Santoro, and Giovanni Viglietta. 2014. The Mutual Visibility Problem for Oblivious Robots. In *Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014*. Carleton University, Ottawa, Canada.

[17] Gabriele Di Stefano. 2022. Mutual visibility in graphs. *Appl. Math. Comput.* 419 (2022), 126850.

[18] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro (Eds.). 2019. *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*. Lecture Notes in Computer Science, Vol. 11340. Springer.

[19] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. 2017. Distributed computing by mobile robots: uniform circle formation. *Distributed Computing* 30 (2017), 413–457. Issue 6.

[20] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro (Eds.). 2019. *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*. LNCS, Vol. 11340. Springer.

[21] Gary L. Miller and John H. Reif. 1991. Parallel Tree Contraction, Part 2: Further Applications. *SIAM J. Comput.* 20, 6 (1991), 1128–1147.

[22] Pavan Poudel, Aisha Aljohani, and Gokarna Sharma. 2021. Fault-tolerant complete visibility for asynchronous robots with lights under one-axis agreement. *Theor. Comput. Sci.* 850 (2021), 116–134.

[23] Pavan Poudel, Gokarna Sharma, and Aisha Aljohani. 2019. Sublinear-time mutual visibility for fat oblivious robots. In *Proceedings of the 20th International Conference on Distributed Computing and Networking, ICDCN 2019*. ACM, 238–247.

[24] Nicola Santoro. 2007. *Design and Analysis of Distributed Algorithms*. John Wiley & Sons.

[25] Gokarna Sharma, Costas Busch, and Supratik Mukhopadhyay. 2015. Mutual Visibility with an Optimal Number of Colors. In *Proc. 11th Int.'l Symp. on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS) (Lecture Notes in Computer Science, Vol. 9536)*. Springer, 196–210.

[26] Gokarna Sharma, Ramachandran Vaidyanathan, and Jerry L. Trahan. 2021. Optimal Randomized Complete Visibility on a Grid for Asynchronous Robots with Lights. *Int. J. Netw. Comput.* 11, 1 (2021), 50–77.

[27] Masafumi Yamashita and Ichiro Suzuki. 2010. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theor. Comput. Sci.* 411, 26-28 (2010), 2433–2453.