Oblivious Permutations on the Plane

Shantanu Das[1], Giuseppe A. Di Luna[2], Paola Flocchini[3], Nicola Santoro[4], Giovanni Viglietta[5], Masafumi Yamashita[6]

Abstract

We consider a distributed system of n identical mobile robots operating in the two dimensional Euclidian plane. As in the previous studies, we consider the robots to be anonymous, oblivious, dis-oriented, and without any communication capabilities, operating based on the Look-Compute-Move model where the next location of a robot depends only on its view of the current configuration. Even in this seemingly weak model, most formation problems which require constructing specific configurations, can be solved quite easily when the robots are fully synchronized with each other. In this paper we introduce and study a new class of problems which, unlike the formation problems so far, cannot always be solved even in the fully synchronous model with atomic and rigid moves. This class of problems requires the robots to permute their locations in the plane. In particular, we are interested in implementing two special types of permutations – permutations without any fixed points and permutations of order n. The former (called Move-All) requires each robot to visit at least two of the initial locations, while the latter (called Visit-All) requires every robot to visit each of the initial locations in a periodic manner. We provide a characterization of the solvability of these problems, showing the main challenges in solving this class of problems for mobile robots. We also provide algorithms for the feasible cases, in particular distinguishing between one-step algorithms (where each configuration must be a permutation of the original configuration) and multi-step algorithms (which allow intermediate configurations). These results open a new research direction in mobile distributed robotics which has not been investigated before.

1 Introduction

The investigation of the computational and complexity issues arising in distributed systems of autonomous mobile robots is an important research topic in distributed computing. This has several applications, teams of robots could be sent to regions inaccessible to humans to perform a variety of tasks such as exploration and data-collection, monitoring, sensing or patrolling. Once deployed, the team of robots must coordinate with each other and perform the tasks autonomously without human intervention; this has motivated the design of distributed algorithms for coordination among the robots to enable them to perform the required tasks.

As a theoretical abstraction, the robots are usually viewed as computational entities modelled as points in a metric space, typically $\mathbb{R}^2$, in which they can move. The robots, identical and outwardly indistinguishable, have the same capabilities and execute the same (deterministic) algorithm. They can see each other, but cannot explicitly communicate with one another. This lack of direct communication capabilities means that the only means of interaction between robots are observations and movements: that is, communication is stigmergic. Each robot operates in "Look-Compute-Move" (LCM) cycles: during a cycle, it observes its surroundings, computes a destination point, and moves to it. Typically, the robots are assumed to have constant-size persistent memory or, more commonly, to be *oblivious*

[1]Aix-Marseille University and LiS Laboratory, shantanu.das@lif.univ-mrs.fr.

[2]DIAG, University of Rome "Sapienza", diluna@diag.uniroma1.it - Part of this work was performed while the author was affiliated with Aix-Marseille University and LiS Laboratory.

[3]University of Ottawa, paola.flocchini@uottawa.ca.

[4]Carleton University, santoro@scs.carleton.ca.

[5]JAIST, johnny@jaist.ac.jp.

[6]Kyushu University, masafumi.ymashita@gmail.com.

having no persistent memory: This paper assumes the latter model where robots in each cycle act only based on the current observation and have no memory of their activities from previous cycles. Further the robots do not have any means of orienting themselves; Each robot observes the location of other robots relative to its own position in the plane and the robots do not share any common coordinate system.

Some typical problems that have been studied in this model include: *gathering* of robots (e.g., [4,8,9]), uniform *dispersal*, *filling* a region with robots, *flocking*, etc. (for a review, see [12]). A generalization of some of these problems is that of *pattern formation*, where the n robots need to move from any initial configuration to a predefined pattern of n points in the plane. This class has been extensively studied (e.g., [1,2,6,13,16,18–21]). A major issue in such formation problems is the amount of symmetry (quantified by the notion of symmetricity [19]) in the starting configuration of robots and in the points of the pattern. In the arbitrary pattern formation problem, the points where the pattern is formed are *relative*, i.e. subject to rotation, translation and scaling of the input pattern. A different line of research is when the points of the pattern are *fixed*, a setting called *embedded pattern* and studied in [3,15].

In some applications, forming a pattern may be the first step of a more complex task requiring coordination between robots. Consider, for example, robots that contain instrumentation for monitoring a site once there, as well as sensors for measurement (e.g., detecting traces of oil or precious metals, radioactivity, etc). If each robot has different sensors, the same site might need to be visited by all robots, and this must be done while still keeping all the sites monitored. A more relaxed version of this task is where each site must be visited by (at least) two robots. This task may be useful even in situations where all the robots contain the same sensors, e.g., if there are faulty sensors and we want to replicate the measurements.

These tasks are instances of a new class of problems quite different from the formation problems as the robots need to rotate among the given points of interests, forming permutations of a given pattern of points. We assume that each robot is initially occupying a point of interest (thus marking that location) and the objective is to permute the robots among these locations periodically. The question is which permutations can be implemented starting from which patterns. We show a big difference between these classes of permutation problems compared to the formation problems studied previously. In particular, we show that even in the *fully synchronous* ($\mathcal{FSYNC}$) model, some of the permutation problems are not solvable, even when starting from configurations that admit a leader. In contrast, any formation problem (including gathering) is easily solvable in $\mathcal{FSYNC}$ when the starting configuration admits a leader.

Note that the permutation problems considered in this paper are *perpetual* tasks requiring continuous visits to the sites by the robots. Unlike the multiple pattern formation problem where robots continuously move from a pattern to the next [6], here the robots perpetually move but only exchanging locations in the same pattern. In particular, we focus on two interesting types of permutations — permutations without fixed points, and permutations of order n (i.e. n-cycles). These give rise to two specific problems (i) Move-All: every site must be visited by at least two robots and every robot has to visit at least two points, and, (ii) Visit-All: every robot must visit each of the points of interest. We provide a characterization of the solvability of these problems showing which patterns make it feasible to solve these problems and under what conditions. To the best of our knowledge, this is the first investigation on these class of problems.

Our Contributions

We distinguish between 1-step and multi-step algorithms; In the former case, we must form the permutations without passing through intermediate configurations, while in the latter case, a fixed number of intermediate configurations are allowed (see definitions in Section 2). We study 1-step and 2-step algorithms for Visit-All and Move-All, distinguishing the case when the robots share a common chirality from the case when they do not. We identify a special class of configurations denoted by $\mathcal{C}_\odot$, that are rotationally symmetric with exactly one robot in the center of symmetry. Such configurations do not always allow permutations without fixed points, thus making it difficult to solve the above problems.

We show that when there is chirality, the sets of initial configurations from which Visit-All and Move-All can be solved, using 1-step algorithms, are the same: that is, all configurations except those in $\mathcal{C}_\odot$ (Section 3). We then show that the characterization remains the same when we consider 2-step algorithms. Moreover, in the case of Visit-All, the solvability does not change even for k-step algorithms for any constant k.

On the other hand, when there is no-chirality, we observe a difference between the solvability of Visit-All and Move-All. Configurations in $\mathcal{C}_\odot$ are clearly still non feasible for both problems. However, for the Move-All problem the class of unsolvable configurations also includes the ones where there exists a symmetry axis with a unique robot on it. On the other hand, the set of initial configurations from which Visit-All is solvable is different: the problem can be solved if and only if in the initial configuration there are no axes of symmetry or if there is a unique symmetry axis that does not contain any robots. Interestingly, also in this case, allowing 2-step algorithms does not change the set of solvable instances.

We then show that, when there is a common chirality and the reference systems of robots are visible (that is, a robot can sense the local coordinate system of the others), then Visit-All (and thus Move-All) is solvable from arbitrary initial configurations, and we provide a universal algorithm for solving the problems. Finally, we show that allowing a single bit of persistent memory per robot and assuming common chirality, it is possible to solve the problems for all initial configurations (Section 6).

2 Model, Definitions and Preliminaries

Robots and scheduler. We consider a set of dimensionless computational entities: the *robots*. Robots operate in the metric space $\mathbb{R}^2$; they are able to look at the environment, perform some local computation, and move to a new destination point. Each robot has its own local coordinate system centred in its position (which may differ in orientation and unit distance from the reference system of other robots). For simplicity of description, we will use a global coordinate system S for analyzing the moves of the robots (robots themselves are unaware of this global system). Robots are *oblivious*: they do not have any persistent memory and thus, they cannot recall any information from previous computations. We indicate the set of robots with $R : \{r_0, r_1, \ldots, r_{n-1}\}$, however the robots themselves are not aware of the numbering assigned to them. All robots are identical and follow the same algorithm. We assume the so-called *Fully-Synchronous Scheduler* ($\mathcal{FSYNC}$). Under this scheduler, time can be seen as divided in discrete fixed length slots called *rounds*. In each round, each robot synchronously performs a *Look-Compute-Move* cycle [12]. During the Look phase a robot takes an instantaneous *snapshot* of the environment, the snapshot is an entire map of the plane containing positions of all the other robots. During the Compute phase a robot performs some local computation to decide a destination point as a function of the aforementioned snapshot as input. Finally, in the Move phase the robot moves to the computed destination point (which may be the same as current location).

Chirality. Robots may or may not share the same *handedness*: in the former case they all agree on the clockwise direction and we say the system has *chirality* [12], in the latter case, robots do not have such an agreement and we say there is *no chirality*.

Configurations. A configuration C is an ordered tuple of points $C = (p_0, p_1, \ldots, p_{n-1})$, where $p_i = C[i]$ is the position of robot r_i in terms of the global coordinate system S. We denote by $Z = (Z_0, Z_1, \ldots Z_{n-1})$ the ordered tuple of coordinate systems where Z_i is the system used by robot r_i. Given a robot r_i located at p_i, we denote with $C \setminus \{r_i\}$ (or sometimes $C \setminus \{p_i\}$), the configuration obtained by removing robot r_i from C. We indicate with C_0 the initial configuration in which the robots start. We denote by $SEC(C)$ the smallest circle that encloses all points in the configuration C.

Symmetricity. Given any configuration C with robots having coordinate systems Z, the symmetricity $\sigma(C, Z) = m$ is the largest integer m such that the robots can be partitioned into classes of size m where robots in the same class have the same view in C. Alternatively, we can define the symmetricity (irrespective of Z) of a configuration as $\rho(C) = m$ where m is largest integer such that $\exists Z : \sigma(C, Z) = m$. For any configuration C, we have $\rho(C) \geq 1$, the configurations with $\rho(C) = 1$ are considered to be asymmetric (these configurations allow to elect a leader among robots). For symmetric configurations with $\rho(C) > 1$, the point o is called the center of symmetry (which coincides with the centroid of C). See Figure 1 for an example of symmetry classes.

We define a special class of configurations denoted by $\mathcal{C}_\odot$. A configuration C is in $\mathcal{C}_\odot$, if and only if $\rho(C) = 1$, and there exists a unique robot r_c (the *central robot*) located at the center of $SEC(C)$ and C has a rotational symmetry around r of degree $n > 1$, that is C can be rotated using as centre r_c of some angle $\theta = \frac{\pi}{n}$ and the resulting configuration is a permutation of C, this is equivalent to say that $\rho(C \setminus \{r_c\}) > 1$. Figure 1 is an example of a configuration in $\mathcal{C}_\odot$.

Permutations and runs. For a permutation $\pi = (\pi(0), \pi(1), \ldots, \pi(n-1))$ of $(0, 1, \ldots, n-1)$, define

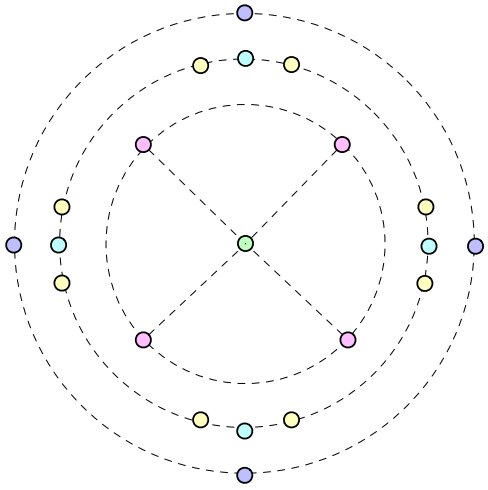


Figure 1: Configuration $C \in \mathcal{C}_{\odot}$, each symmetry class is coloured in a different way. Notice that the central robot r_c is the only one with an unique view (recall that the central robot is the green one, the robot positioned on the centroid of C). If removed we get a configuration $C \setminus \{r_c\}$, with $\rho(C \setminus \{r_c\}) = 4$, notice also that the yellow class has cardinality 8.

$\pi(C) = (p_{\pi(0)}, p_{\pi(1)}, \ldots, p_{\pi(n-1)})$. We denote the special set of permutations $\Pi_2 = \{\pi : \pi(i) \neq i, \forall i : 0 \leq i \leq n-1\}$, the set of permutations with no fixed points and $\Pi_n = \{\pi : \pi^j(i) = i <=> nk = j$ for some $k \in \mathbb{N}\}$ where π^j indicates that we apply permutation π j times, the set of cyclic permutations of order n. Let $\Pi(C)$ be the set of all permutations of C, sometimes with a small abuse of notation we write $C' \in \Pi(C)$ to indicate that $C' = \pi(C)$ for some $\pi \in \Pi(C)$.

Given an algorithm $\mathcal{A}$ and an initial configuration C_0 we denote any execution of algorithm A, starting with configuration C_0 as the run $\mathcal{R}_{\mathcal{A},C_0} : (C_0, C_1, C_2, \ldots)$, the infinite ordered sequence of configurations, where C_j is produced during round j of the execution.

Problem Definitions

We will study the following two problems:

- Move-All: An algorithm $\mathcal{A}$ is a 1-step solution algorithm for the Move-All problem, if every possible run of the algorithm $\mathcal{R}_{\mathcal{A},C_0} : (C_0, C_1, C_2, \ldots)$ is such that: $C_i = \pi^i(C_0)$ for some $\pi \in \Pi_2$. Intuitively, every configuration is a permutation of C_0 and in any two consecutive configurations, the position of each robot is different. As an extension for any $k \in \mathbb{N}^+$, a k-step solution requires that $C_{i \cdot k} = \pi^i(C_0)$ where $\pi \in \Pi_2$. (There is no constraint on the intermediate configurations C_j where k does not divide j.)

- Visit-All: An algorithm $\mathcal{A}$ is a 1-step solution algorithm for the Visit-All problem, if every possible run of the algorithm $\mathcal{R}_{\mathcal{A},C_0} : (C_0, C_1, C_2, \ldots)$ is such that: $C_i = \pi^i(C_0)$ for some $\pi \in \Pi_n$. Intuitively, every configuration is a permutation of C_0 and in every n consecutive configurations, every robot visit every location $p_i \in C_0$. We can similarly define a k-step solution for the problem where $C_{i \cdot k} = \pi^i(C_0)$ for some $\pi \in \Pi_n$.

Since $\Pi_n \subset \Pi_2$, it follows that any solution for Visit-All is also a solution to the Move-All problem.

Oblivious Permutations

Note that k-step solutions of Move-All and Visit-All specify that we must have a permutation of the initial configuration C_0 every k rounds. However, no constraint is given on the other *intermediate*

configurations. Interestingly, when robots are oblivious the previous definitions imply a stronger version of the problem in which each configuration C_{j+k} has to be the permutation of configuration C_j that appeared k rounds ago.

Lemma 1. *Let $\mathcal{A}$ be a k-step algorithm solving* Move-All *(or* Visit-All*), and let $\mathcal{R}_{\mathcal{A},C_0} : (C_0, C_1, C_2, \ldots)$ be any run of $\mathcal{A}$ starting from C_0. For each $j \in \mathbb{N}$ we have that $C_{j+k} = \pi(C_j)$ for some $\pi \in \Pi(C_j)$.*

Proof. We prove the lemma for Move-All, the extension to Visit-All is analogous and immediate. If $j = t \cdot k$ for some $t \in \mathbb{N}$ then the lemma follows from the problem definition. Thus let us consider a configuration C_j such that $j \neq t \cdot k$ for all $t \in \mathbb{N}$ and let us suppose $C_{j+k} \neq \pi(C_j)$. We observe that $\mathcal{R}_{\mathcal{A},C_j}$ (that is a run of $\mathcal{A}$ starting from C_j) is equal to the suffix of $\mathcal{R}_{\mathcal{A},C_0}$ starting from C_j. This is due to the obliviousness of the robot, the fact that the algorithm is deterministic and the synchronous scheduler: starting from a certain configuration and an assignment of local reference system will generate a fixed sequence of configurations. However in $\mathcal{R}_{\mathcal{A},C_j}$ we must have that $C_{j+k} = \pi(C_j)$ for some $\pi \in \Pi_2(C_j)$, otherwise $\mathcal{A}$ is not a correct algorithm for Move-All. □

3 Oblivious Robots with Chirality

In this section we consider robots with the same handedness (e.g., same clockwise orientation).

3.1 1-Step Algorithms

We first consider 1-step algorithms, and show that Move-All and Visit-All are solvable if the initial configuration C_0 is not in $\mathcal{C}_\odot$.

Intuition behind the solution algorithms. The underlining idea of our solution algorithms is to first make robots agree on a cyclic ordering of the robots, and then permute their positions according to this ordering. This algorithm is shown in Algorithm 1 and the ordering procedure is shown in Algorithm 2. When the centroid c of configuration C_0 does not contain any robot, we compute a cyclic ordering on the robots by taking the half-line passing through c and one of the robots closest to c and rotating it w.r.t. point c; the robots are listed in the order the line hits them. We can show that the ordering computed by any robot is a rotation of that computed by another robot (See Figure 2 for example). The only issue is when there is a robot positioned in the centroid. In this case, the robots compute a unique total order on the robots; this is always possible since $C_0 \notin \mathcal{C}_\odot$, which implies that C_0 is asymmetric and admits a total ordering.

From the above observations, It is immediate that the algorithm solves Visit-All, take a robot r, w.l.o.g. in position p_i, during n activations, the robot moves through all the robot positions in the computed cyclic order, returning back to p_i; thus, it has visited every point in C_0.

Algorithm 1 Visit-All Algorithm using a cyclic order.

1: Compute a cyclic order $(p_0, p_1, \ldots, p_{n-1})$ on C using Order(C).
2: If my position is p_i, set **destination** as $p_{(i+1) \mod n}$.

Theorem 1. *In systems with chirality, there exists a 1-step algorithm that solves* Visit-All *from any initial configuration $C_0 \notin \mathcal{C}_\odot$.*

Proof. The proof is constructive; in fact, we show a 1-step algorithm that solves Visit-All. Each robot obtains a cyclic order by executing Algorithm 2. To prove that Visit-All can be solved, it is sufficient to show that the cyclic orders are the same for all robots.

The order is decided by Procedure Order. We distinguish two cases, when the centroid c of C_0 is not in C_0, and when it is in C_0.

- Case of $c \notin C_0$: Let us consider by contradiction that for two different robots, w.l.o.g. r_1, r_2, there are two different cyclic orderings. Let L_1 be the list given by Order of robot r_1 and let L_2 be the

Algorithm 2 ORDER algorithm with Chirality.

1: **procedure** INSERT(list, Configuration C) ▷ if this procedure is called then $\rho(C_0) = 1$.
2: Let *order* be a total order of robots in C with the centroid c as last element, and let p the position that precedes c in *order*.
3: insert c in list after position p
4: **return** list

5: **procedure** POLAR(position p, reference r, Configuration C)
6: POLAR takes two points p, r, and a configuration C.
7: POLAR returns the polar coordinates (d, θ) of p in a reference system that is centered in the centroid of C and has as reference direction the segment between the centroid and point r.
8: **return** (d, θ)

9: **procedure** NEXT(position r, Configuration C)
10: **if** $\exists p \in C$ such that $(d', 0)$ =POLAR(p,r,C) with $d' > ||r||$ **then**
11: **return** p
12: **else**
13: Let $p \in C$ be such that (d, θ) =POLAR(p,r,C) has the minimum $\theta > 0$ and the minimum d among all the polar coordinates of robots in C.
14: **return** p

15: **procedure** ORDER(Configuration C)
16: c=centroid of C
17: list=EmptyList()
18: r_1=pick one robot position in C with minimum non zero distance from c.
19: list.Append(r_1)
20: $A = C \setminus \{r_1, c\}$
21: **while** $A \neq \emptyset$ **do**
22: r_2=NEXT(list.LastElement(),C)
23: list.Append(r_2)
24: $A = A \setminus \{r_2\}$
25: **if** $c \in C$ **then**
26: list=INSERTCENTROID(list,C)
27: return list

list given by ORDER of robot r_2. Being L_1 and L_2 different, then there exist two indices j_1, j_2 such that $L_1[(j_1) \mod n] = L_2[(j_2) \mod n]$ and $L_1[(j_1 + 1) \mod n] \neq L_2[(j_2 + 1) \mod n]$. However, this implies that $\exists p_j \in C_0$, such that NEXT(p_j, C_0) gives two different results according to the reference system of the callee, that is impossible: the centroid is the same for each one of them, moreover robots agree on the same handedness therefore the angles grows counterclockwise for all of them.

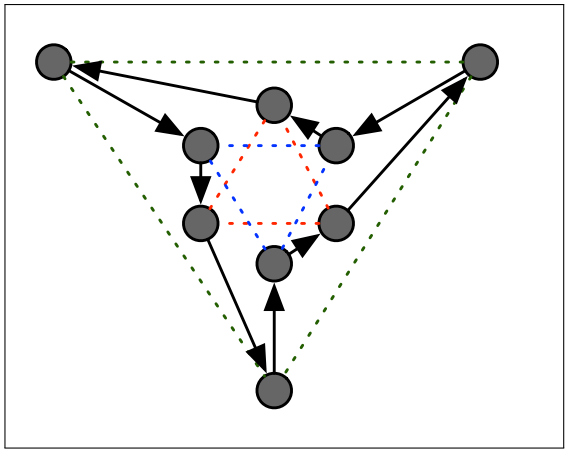
- Case of $c \in C_0$: First of all, note that the argument of the previous case implies that the *list* obtained before executing Line 26 of Algorithm 2 represents a common cyclic order on $C_0 \setminus \{c\}$ shared by all robots. It remains to show that, when executing Line 26, all robots insert the centroid c before the same position p in *list*. We have by assumption that $C_0 \notin \mathcal{C}_\odot$, thus if $c \in C_0$ then there exists a total order on which all robots agree: by definition we either have $\rho(C_0 \setminus \{c\}) = 1$ (and the total order is trivially enforced) or that there is an unique symmetry axis of C; in the latter case the presence of chirality ensures such total order. It is easy to see that from this total order we can obtain a total order on which c is the last element, and on which all robots agree, let it be *order*. Therefore, being *order* common to all robots, it is immediate that everyone inserts c before the same position p when executing Line 26.

□

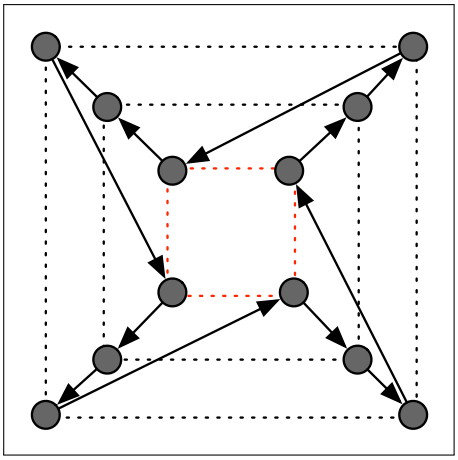
We now show that, when $C_0 \in \mathcal{C}_\odot$ MOVE-ALL (and thus VISIT-ALL) is unsolvable:

Theorem 2. *If $C_0 \in \mathcal{C}_\odot$ there exists no 1-step algorithm that solves* MOVE-ALL*, even when the robots agree on a common chirality.*

Proof. In any configuration in $\mathcal{C}_\odot$, we can always assign the reference systems in such a way that each robot, except the central robot c, has at least one analogous with a symmetric view. This derives directly



(a) An example of cyclic order induced by the ORDER algorithm.



(b) Another example of cyclic order induced by the ORDER algorithm.

Figure 2: ORDER algorithm: Examples.

from the definition of $\mathcal{C}_\odot$. It is immediate to see that it is impossible to elect a unique robot to move to the center of C_0. An example is given in Figure 3, where if one robot of the pink class moves to the center of C_0, then every robot in the pink class does the same. This implies that, in the next round, it is impossible to form any $C_1 = \pi(C_0)$ for some $\pi \in \Pi(C_0)$ with a different central robot. □

Note that Theorem 1 implies that MOVE-ALL is solvable under the same assumptions of the theorem (recall that if we satisfy the VISIT-ALL specification, we satisfy also MOVE-ALL specification). Moreover, for the same reason, Theorem 2 implies that VISIT-ALL is unsolvable.

We can summarize the results of this section as follows:

Theorem 3. *In systems with chirality,* MOVE-ALL *and* VISIT-ALL *can be solved in 1-step if and only if* $C_0 \notin \mathcal{C}_\odot$.

3.2 2-step Algorithms

In light of Theorem 3, one may wonder what happens when multiple steps are allowed. In this section we show that allowing an intermediate step to reach the goal does not bring any advantages. We first introduce a technical lemma. Intuitively, the result is based on the observation that it is impossible to replace the central robot by another robot in 1-step. Thus the intermediate configuration must be a configuration $C_1 \notin \mathcal{C}_\odot$.

Lemma 2. *Let* $\mathcal{A}$ *be a 2-step algorithm that solves* MOVE-ALL. *Starting from configuration* $C_0 \in \mathcal{C}_\odot$, *algorithm* $\mathcal{A}$ *cannot generate a run* $\mathcal{R}_{\mathcal{A},C_0} : (C_0, C_1, C_2, C_3, \ldots)$ *where* $C_1 \in \mathcal{C}_\odot$.

Proof. Being $\mathcal{A}$ a 2-step algorithm we have $C_2 = \pi(C_0)$ for some $\pi \in \Pi(C_0)$, and being a solution for MOVE-ALL we have that the central robot, let it be r_c, in C_0 has to be different from the central robot in C_2. By assumption we have $C_1 \in \mathcal{C}_\odot$, we will show that r_c is also the central robot in C_1. Let us suppose the contrary, and let $r_x \neq r_c$ be the central robot in C_1. Note that, by definition of $\mathcal{C}_\odot$, robot r_x is the only robot that has a view different from all the others in configuration C_1. This is equivalent to say that, in a configuration obtained by removing r_c from C_0, robot r_x may move to a position in such a way to make its view unique and breaking the symmetry of the configuration. This directly contradicts the fact that $C_0 \in \mathcal{C}_\odot$, in C_0 the only robot with a unique view is r_c and it is the only robot that can break the symmetry by moving. Therefore, the central robot in C_1 has to be r_c. By using the same argument, we have that also the central robot of C_2 has to be r_c, which contradicts the correctness of $\mathcal{A}$. □

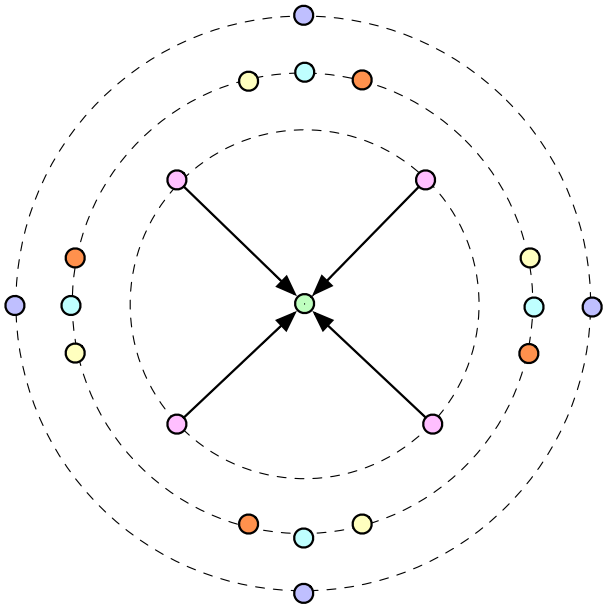


Figure 3: Configuration $C_0 \in \mathcal{C}_\odot$ where it is impossible to solve MOVE-ALL with a 1-step algorithm.

Based on the above result, we can show that is impossible to solve MOVE-ALL from a configuration $C_0 \in \mathcal{C}_\odot$, even if the system has chirality. The informal idea here is that the central robot r_c in configuration C_0 needs to move away from the center to form the intermediate configuration C_1. However, in any 2-step algorithm, C_2 must be a permutation of C_0, with a different robot r' in the center. Now, following the same algorithm, robot r' would move away from the center to form the next configuration C_3. By choosing the coordinate systems of robots r_c and r' in an appropriate way, the adversary can ensure that C_3 would not be a permutation of C_1, thus violating the conditions in Lemma 1. The above reasoning is formalised in the following theorem:

Theorem 4. *There exist no* 2*-step algorithm that solves* MOVE-ALL *from a configuration* $C_0 \in \mathcal{C}_\odot$*, even if the system has chirality.*

Proof. Consider a configuration $C_0 \in \mathcal{C}_\odot$ where all robots but the central are vertices of a regular polygon, as a reference see Figure 4. Let us assume that there exists a 2-step algorithm $\mathcal{A}$ that solves MOVE-ALL starting from C_0. Let r_c be the central robot in C_0, with reference system Z_{r_c}, and let $NC : \{r_1, \ldots, r_{n-1}\}$ be the set of the other robots. There are two possible behaviours of $\mathcal{A}$ according to which a robot moves when $\mathcal{A}$ starts from configuration C_0:

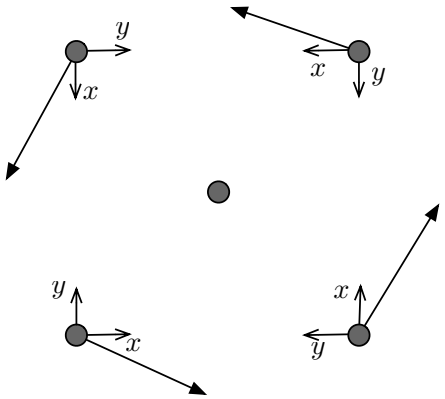


Figure 4: Case in which algorithm A makes only (and all) robots in NC move: the resulting configuration will be still in $\mathcal{C}_\odot$.

- Robot r_c moves, and possibly also robots in NC. Being $\mathcal{A}$ a 2-step algorithm, starting from configuration $C_0 \in \mathcal{C}_\odot$, algorithm $\mathcal{A}$ generates a run $(C_0, C_1, C_2, C_3, \ldots)$ where $C_x \in \Pi(C_0)$ when x is even, and $C_x \notin \mathcal{C}_\odot$ when x is odd (see Lemma 2). Let us now assume that the reference system of any $r \in NC$ is obtained by rotating the reference system of robot r_c by $\frac{\pi}{2}$. Since $\mathcal{A}$ must solve Move-All, we have that in C_2 the robot in the center of symmetry cannot be r_c, let it be r. Note that r will move to a destination point that is different from the one r_c moved starting from C_0, which means that configuration $C_3 \notin \Pi(C_1)$. However, this leads to a contradiction, see Lemma 1.
- Only robots in $NC : \{r_1, \ldots, r_{n-1}\}$ move. First note that all robots in NC have symmetric views. Therefore, it is not possible to make only one of them move, everyone has to move. With this is mind is easy to see that, for any possible movement option of robots in NC, there exists an initial arrangement of local coordinate systems such that the resulting configuration will be a rotation and a scaling of the initial configuration, see Figure 4 for an example. Therefore, configuration C_1 is still in $\mathcal{C}_\odot$, by Lemma 2 this implies that $\mathcal{A}$ is not correct.

Since $\mathcal{A}$ is not correct in any of the previous cases, we have that the existence of $\mathcal{A}$ is impossible. □

Interestingly, when we consider Visit-All we can prove a stronger impossibility result that includes algorithms that could depend on the number of robots n, and it shows that such algorithms cannot solve Visit-All as long as the number of steps is $\mathcal{O}(1)$.

Theorem 5. *There exists no k-step algorithm for* Visit-All*, where $k = \mathcal{O}(1)$ with respect to the size of n. The above holds even if the system has chirality*

Proof. Let us consider a starting configuration $C_0 \in \mathcal{C}_\odot$ such that robot r_c is central, and all the robots $\{r_1, r_2, \ldots, r_{n-1}\}$ that are not central are positioned in the vertices of a regular $n-1$-gon. Moreover, let us assume that the reference systems of robots $r_1, \ldots, r_{n-1}$ are pairwise different. The proof is by contradiction. Let $\mathcal{A}$ be a Visit-All algorithm that uses k-step. The first observation is that any algorithm solving Visit-All has to ensure that, for any robot r_j, there exists a configuration $C_j \in \Pi(C_0)$ that will be reached by the algorithm and such that r_j occupies the central position in C_j, let this position be p_c. The second observation is that from a configuration in $\mathcal{C}_\odot$, algorithm $\mathcal{A}$ has to reach a configuration $C \notin \mathcal{C}_\odot$ where the robot in position p_c moved (see Lemma 2).

The two above observations imply that the run of $\mathcal{A}$ that starts from configuration C_0 contains at least $n-1$ different configurations not in $\mathcal{C}_\odot$, one for each different robot in $\{r_1, r_2, \ldots, r_{n-1}\}$. This is immediate observing that from each C_j the successive configuration in the run will be a specific N_j different from the others $N_{i \neq j}$: starting from C_j robot r_j occupying p_c has to move in a position different from the one where $r_{i \neq j}$ in configuration C_i moved, this is ensured by the fact that the reference system of r_j is different from the ones of other robots.

It is also easy to see that each N_j could be a starting configuration for algorithm $\mathcal{A}$, thus algorithm $\mathcal{A}$ starting from N_j generates a run $\mathcal{R}_{\mathcal{A},N_j} : (X_0, X_1, X_2, \ldots)$ where permutations of N_j appear at most every k-steps (this property is due to the fact that $\mathcal{A}$ is a k-step algorithm). Let call this fact (F1).

Being the algorithm for oblivious robots, if N_j appears in $\mathcal{R}_{\mathcal{A},C_0}$ at round r, then, from r on, we have $\mathcal{R}_{\mathcal{A},C_0}[r+t] = \mathcal{R}_{\mathcal{A},N_j}[t]$ for any $t > 0$. Let this be fact (F2). Let r^* the first round when, in $\mathcal{R}_{\mathcal{A},C_0}$ every N_j, with $j \in [1, n-1]$, appeared. Let us consider the set X of configurations that appears in $\mathcal{R}_{\mathcal{A},C_0}$ between round r^*+1 and r^*+k, clearly we have $|X| \leq k$. At the same time, all configurations N_* appearing before r^*-1 have to appear at least once in the interval $\mathcal{R}_{\mathcal{A},C_0}[r^*+1, r^*+k]$, this comes directly from (F1) and (F2). By the pigeonhole principle, this is impossible: there are at least $n-1$ such configurations and $|X| \leq k < n-1$. This contradicts the existence of $\mathcal{A}$. □

4 Oblivious Robots without Chirality

In this section we consider robots that do not share the same handedness. Interestingly, the absence of chirality changes the condition for solvability of Move-All and Visit-All, showing the difference between these two problems. This is due to the fact that in systems without chirality, the configuration of robots may have mirror symmetry, in addition to rotational symmetry as in the previous section.

4.1 Move-All

The following theorem illustrates the configurations for which the Move-All problem is unsolvable.

Theorem 6. *In systems without chirality* Move-All *is unsolvable in* 1*-step starting from any configuration $C_0 \in \mathcal{C}_\odot$, as well as from any configuration that has a symmetry axis containing exactly one robot.*

Proof. Unsolvability from $C_0 \in \mathcal{C}_\odot$ follows from Theorem 2. Consider then an initial configuration C_0 such that in C_0 there exists a symmetry axis containing only one robot r, and let $\mathcal{A}$ be a solution algorithm for Move-All for this scenario.

In order to satisfy the Move-All specification, there must be a configuration $C_1 \in \mathcal{R}_{\mathcal{A},C_0}$ following C_0, and in C_1 a robot $r' \neq r$ must be in the position of robot r in C_0; notice that if is not the case then $C_1 \notin \Pi(C_0)$ or r did not move, in both cases Move-All specification has been violated. However, in C_0 there exists a robot r'', symmetric to r' with respect to the axis containing robot r, that has the same view of r', this comes directly from the definition of symmetry axis. Now, if in C_0 robot r' decides as destination the position of r, also r'' does the same, this implies that $C_1 \notin \Pi(C_0)$. See the example of Figure 5. This contradiction disproves the correctness of $\mathcal{A}$. □

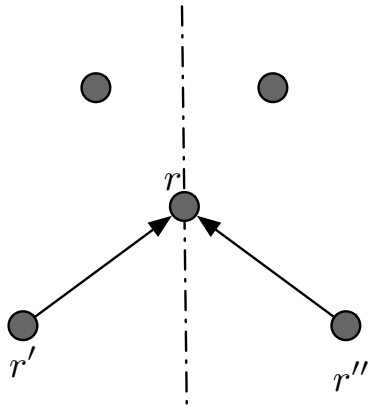


Figure 5: Initial configuration with an unique robot r on the symmetry axis. If robot r' moves in the position of r, then r'' does the same.

We now consider the solutions to the Move-All problem for the feasible instances. When the initial configuration has no rotational symmetry nor any symmetry axes, then the robots can agree on a common chirality and the algorithms from the previous section can be applied. If the configuration has a central symmetry but no axis of symmetry, then the symmetricity must be even (since $C_0 \notin \mathcal{C}_\odot$, there

is no robot at the center); In this case, each robot can be paired to its counterpart on the opposite end of the center, and the paired robots can swap positions (see Figure 6).

Thus the only remaining configurations are those with an axis of symmetry. For such configurations, it is possible to partition the robots in three disjoint subsets, and it make them move as follows: (see also Figure 8)

- For the robots located on a symmetry axis, there exists a unique cyclic order on these robots. Robots on the axis are permuted according to this ordering.
- The second subset contains robots that are the closest to the symmetry axis. These robots switch positions pairwise, and each one switching with its symmetric robot w.r.t. the closest axis.
- The last subset consists of robots that are equidistant from two symmetry axes. Also in this case robots switch positions pairwise, and each one switches position with its symmetric w.r.t. the centroid c of configuration C_0.

For all the configurations excluded by Theorem 6, Move-All can be solved using the above approach. This algorithm is formally presented in Algorithm 3.

Algorithm 3 Move-All 1-step Algorithm when $C_0 \notin \mathcal{C}_\odot$ and C_0 does not have a symmetry axis containing only one robot.

```
1: if C is central symmetric then
2:     set destination as position p, where p is the symmetric of my position with respect to the center of C.
3: else if C does not have symmetry axes then
4:     Set your clockwise orientation using a deterministic algorithm with input C.
5:     Execute Algorithm 1 of Section 3.1 using the Order procedure of Algorithm 2 of Section 3.1.
6: else
7:     if My position is on a symmetry axis A then
8:         Deterministically order the positions on A, obtaining a cylic order p_0, p_1, ..., p_{t-Compute1}.
9:         if My position is p_i then
10:            set destination as p_{(i+1) mod t}.
11:    else if There is an unique axis of symmetry A closest to my position then
12:        set destination as position p symmetric to my position with respect to A.
13:    else
14:        set destination as position p symmetric to my position with respect to the center of C.
```

Theorem 7. *If $C_0 \notin \mathcal{C}_\odot$ and C_0 does not have a symmetry axis containing exactly one robot, then* Move-All *is solvable in* 1*-step even when the system does not have chirality.*

Proof. The pseudocode is in Algorithm 3. Let us consider an execution starting from a configuration C_0 that respects the assumptions of the theorem.

- If in C_0 there is a central symmetry (see Figure 6) then Lines 1-2 are executed. By definition of central symmetric it is easy to see that the next configuration belongs to $\Pi(C_0)$ and that everyone has moved (e.g., see the arrows in Figure 6).
- If C_0 has no central symmetry and there are no axes of symmetry (see an example in Figure 7), then the branch at line 3 is executed. If $\rho(C_0) = 1$ there exists a total order among robots, thus it is possible to apply the algorithms of Section 3. So let us examine the case when $\rho(C_0) \geq 2$. The absence of a symmetry axis implies that robots can agree on a common notion of clockwise direction using configuration C_0. Once they have a common clockwise notion they solve Move-All using the algorithms of Section 3. An example when $\rho(C_0) > 1$ is shown in Figure 7.

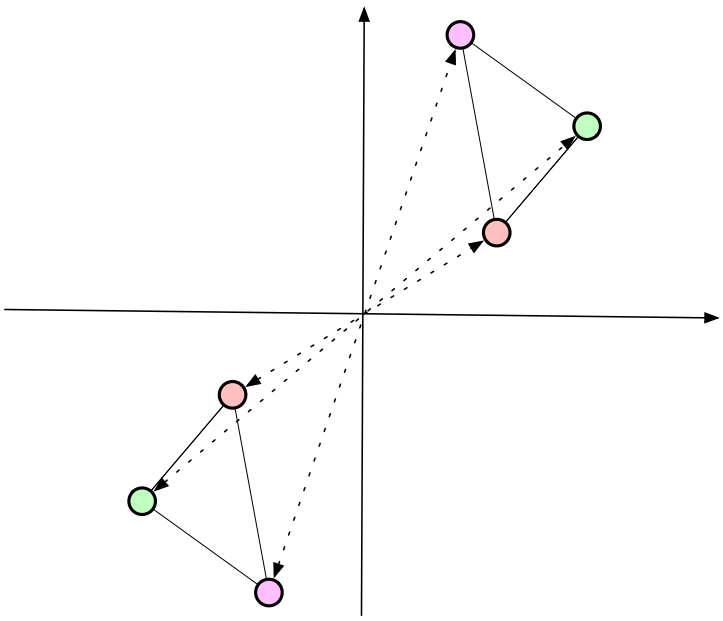


Figure 6: Central symmetric configuration: each robot swaps position with the symmetric obtained by a rotation of π radians with respect to the center. Recall that a configuration C has a central symmetry if once rotated around the centroid of π radians we get a permutation of C.

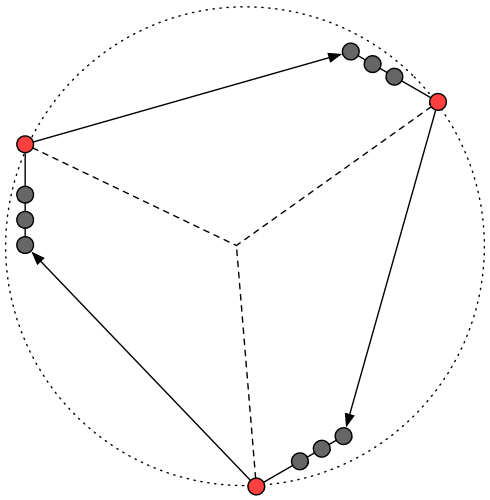


Figure 7: Configuration C_0 that has no central symmetry, no symmetry axis and where $\rho(C_0) = 3$. Notice that robots can agree on a clockwise direction, see arrows.

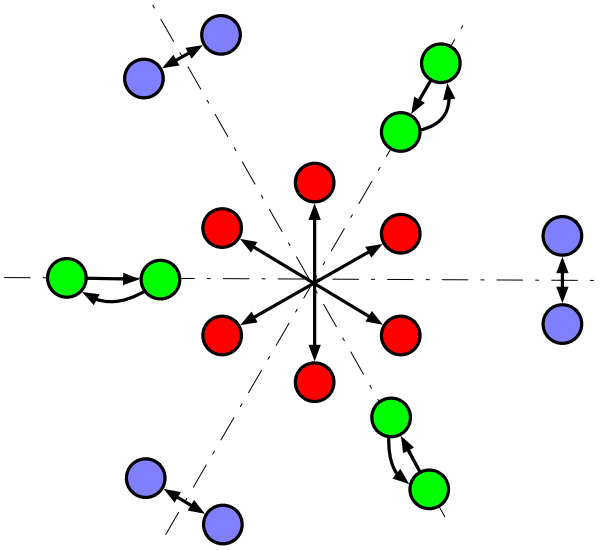


Figure 8: Configuration C_0 that has no central symmetry and there exists no symmetry axis containing an unique robot. The arrows indicate the swap of robots, we have 3 kinds of behaviour: (1) robots on the axis agree on a cyclic order, green in the figure; (2) robots that are equidistant from two axes swap position with the robots that are rotationally symmetric with respect to center of C_0, red in the figure; (3) robots that are closest to an axis of symmetry swap position with the symmetric with respect to that axis, blue in the figure.

- If there is no central symmetry in C_0 and each symmetry axis contains no robots or at least 2 (see Figure 8), then we have 3 possible behaviours according to the position of the robots.
  - If a robot is on a symmetry axis A, it executes the branch at line 7. Robots on A are able to agree on a cyclic order between them. Let us suppose the contrary, then there must exists a second symmetry axis A' that is perpendicular to A, which contradicts the fact that C_0 has no central symmetry. Thus robots on A can be cyclically ordered. It is immediate that if robots on A permute according to this order, then the Move-All problem, restricted to the subset of robots on A, is correctly solved.
  - If there exists a unique axis A closest to a robot r, then the robot r' symmetric to r with respect to A is unique and properly defined, and these two robots are able to swap position. Thus Move-All it is also solved correctly for this subset.
  - If there does not exists a unique axis A closest to r and r is not on a symmetry axis, then we will show that exists a unique robot r' symmetric to r with respect to the center of configuration C_0. Let A and A' be the two axes from which r is equidistant. First, we observe that r is on the bisector segment of the angle θ between two axes of symmetry. Let B be this segment. Second, we observe that due to the fact that A and A' are symmetry axes of C_0, we have that 2π is divided by θ, let us say k times, and that there are k rotations of B around the center of C_0. This implies that there exists a unique robot r' (see Figure 8 as an example where $\theta = \frac{\pi}{3}$). Thus r and r' can swap position safely. Therefore, Move-All it is solved correctly for this last subset.

  Since Move-All is correctly solved for all three cases, and each robot in C_0 belongs to one of the sets considered in the above cases, then Move-All is correctly solved.

□

To summarize, we have the following characterization for solvability of Move-All without chirality:

Theorem 8. *In systems without chirality,* Move-All *is solvable in* 1*-step if and only if* $C_0 \notin \mathcal{C}_\odot$ *and* C_0 *does not have a symmetry axis containing exactly one robot.*

4.2 Visit-All

The Visit-All problem differs from Move-All only when $n > 2$, so we will assume in this section that $n \geq 3$. We will show that Visit-All is solvable without chirality if (i) $C_0 \notin \mathcal{C}_\odot$ and (ii) C_0 does not have symmetry axes, or there is a unique axis of symmetry that does not intersect any point of C_0. The main idea of the algorithm is the following. When C_0 does not have a symmetry axis: then it is possible to agree on a common notion of clockwise direction. Once this is done Algorithm 2 can be used. So we consider the case when C_0 has a unique axis of symmetry that does not intersect any point of C_0: we partition C_0 in two sets C' and C'', containing robots from the two sides of the axis of symmetry. In each of these sets it is possible to agree on a total order of the points (recall that the symmetry axis is unique). Let $order' : [p'_0, p'_1, \ldots, p'_{\frac{n-1}{2}}]$ be the order on C' and $order'' : [p''_0, p''_1, \ldots, p''_{\frac{n-1}{2}}]$ be the analogous for C''. We obtain a cyclic order on C_0 by having element p''_0 following $p'_{\frac{n-1}{2}}$, and, in a symmetric way, p'_0 follows $p''_{\frac{n-1}{2}}$ (see Figure 9).

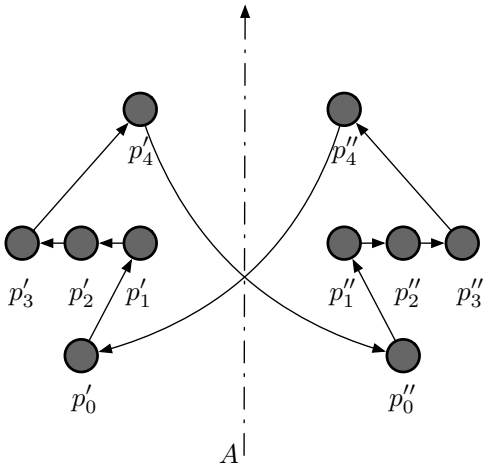


Figure 9: Configuration C_0 with a unique symmetry axis A and no robots intersecting A. The arrow on axis A indicate the direction on which robots agree. The arrows among configuration points indicate the order induced by Algorithm 4.

The corresponding ordering pseudocode is defined in Algorithm 4.

Algorithm 4 Order algorithm without Chirality.

```
1: procedure Order(Configuration C)
2:     if C does not have a symmetry axis then
3:         Set your clockwise orientation using a deterministic algorithm with input C.
4:         return the output of Order procedure of Algorithm 2 of Section 3.1.
5:     else                    ▷ In this case there is a unique symmetry axis that does not intersect any point of C.
6:         Let A be the unique symmetry axis of C.
7:         Let C' and C'' be partitions of C such that C'' is the symmetric of C' w.r.t. A.
8:         Let order' be a total order of robots in C'
9:         Let order'' be a total order of robots in C''
10:        Let list be a list obtained by ordinately appending to order' the elements of order''
11:        return list
```

The correctness of Algorithm 4 is shown in the following theorem.

Theorem 9. *When $n > 2$ and robots do not agree on a common handedness,* Visit-All *is solvable in 1-step, with initial configuration $C_0 \notin \mathcal{C}_\odot$, if one of the following holds:*

- *There are no symmetry axes in C_0*
- *There exists a unique symmetry axis of C_0 and no point of C_0 intersects the axis.*

Proof. Algorithm 4 solves the problem. We show its correctness case by case:

- There are no symmetry axes in C_0: If $\rho(C_0) = 1$, then it is obvious that the robots can agree on a common notion of clockwise direction, and in such a case the order returned at Line 4 is correct by Lemma 1. In case $\rho(C_0) > 2$ it is possible to agree on a common notion of clockwise direction (Line 3) as follows. Since there exist an ordering among the classes constituting C_0, let H be the "highest" class in this order (see red points in Figure 7 as reference). Take a line L passing through the centroid c of C_0 and a point p in H. Since L is not an axis of symmetry, we have that the half-planes defined by L are different and not symmetric w.r.t. to L. Specifically, it is possible to order them using p by scanning the plane starting from p in the local clockwise and counter-clockwise directions using c as center and stopping at the first asymmetry. Therefore, the half-planes can be used to define a common clockwise direction, e.g., the one going from the lowest half-plane to the highest. Note that this procedure gives the same order for any choice of $p \in H$. Once there is an agreement on clockwise direction the correctness derives from the one of Algorithm 2 (see Lemma 1).
- There exists a unique symmetry axis A of C_0 and no point of C_0 intersects A: this case starts at Line 5. Let C' and C'' be partitions of C_0 such that C'' is the symmetric of C' w.r.t. A. First of all, we show that robots can agree on a total order on points in C': given the axis A they can agree on an orientation of the axis (suppose the contrary, then there should exists a second axis intersecting A, that is excluded by hypothesis). We order the robots using the coordinates on A from lowest to highest (robots with the same coordinate are ordered by their distance from the axis). Let this order be $order'$. The analogous can be done for C'', let this order be $order''$. An example is in Figure 9. We merge these two orders by taking the first point in $order''$, let it be p''_0, the last point in $order'$, let it be $p'_{\frac{n-1}{2}}$, and by creating a common cyclic order on which p''_0 follows $p'_{\frac{n-1}{2}}$. Symmetrically, the first point p'_0 in $order'$ follows the last point $p''_{\frac{n-1}{2}}$ in $order''$.

□

Interestingly, without chirality, Visit-All is not solvable if the assumptions of Th. 9 do not hold:

Theorem 10. *When $n > 2$ and there is no chirality, there exists no algorithm that solves* Visit-All *in 1-step from an initial configuration C_0 if one of the following holds:*

- $C_0 \in \mathcal{C}_\odot$
- *There exists a symmetry axis of C_0 intersecting a proper non-empty subset of C_0.*
- *There are at least two symmetry axes of C_0.*

Proof. We prove the impossibility case by case:

- $C_0 \in \mathcal{C}_\odot$: this case derives directly from Theorem 2.
- There exists a symmetry axis A in C_0 and A intersects a proper non-empty subset of C_0: to solve Visit-All we must have that a robot r outside of axis A eventually moves to a position on the axis, let such a position be p. However, being A a symmetry axis, we must have a robot r' symmetric to r w.r.t. A. Robot r' also moves to position p. Since both robots move to point p we reach a configuration that is not in $\Pi(C_0)$.
- There are at least two symmetry axes in C_0: let A and B be these axes. Let P, P_A, P_B, P_{AB} be the portions of the plane defined by the axes, see Figure 10. We can assume that no point in C_0 intersects the axis (otherwise we are in the previous case). Let r be a robot in P, and let r_A, r_B, r_{AB} be the symmetric robots with respect to axis A, B and both axes (see always Figure 10 for a reference), and let p, p_A, p_B, p_{AB} be their respective positions in C_0. Let us suppose, by contradiction, that there exists an algorithm $\mathcal{A}$ solving Visit-All on C_0, and let $\mathcal{R}_{\mathcal{A},C_0} : (C_0, C_1, C_2, \ldots)$ be the corresponding run. We examine the path of robot r among the points in C_0 in the run $\mathcal{R}_{\mathcal{A},C_0} : (C_0, C_1, C_2, \ldots)$. Let us suppose, w.l.o.g., that in C_1 robot r moves to a point in P_A (see the blue dotted path in the Figure 10). It is obvious that the local reference systems

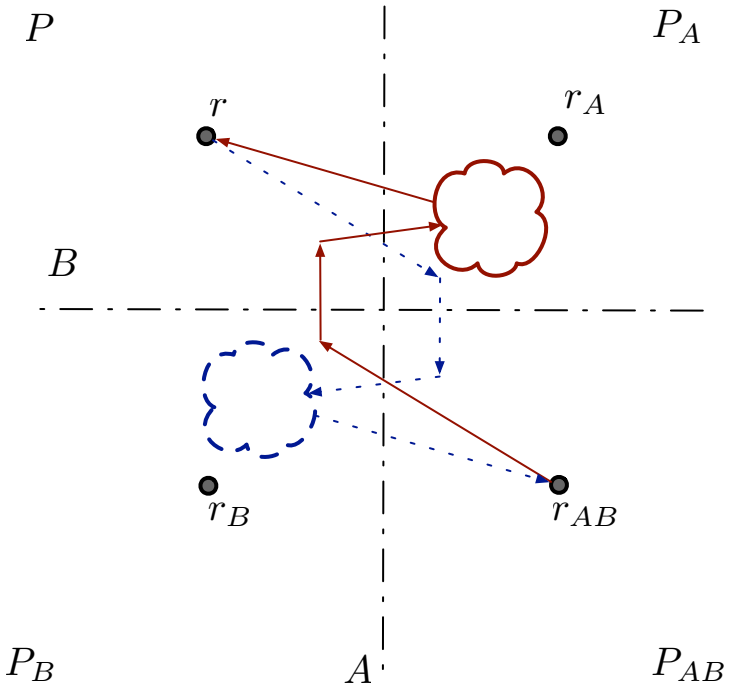


Figure 10: Two symmetry axes in C_0 and no chirality.

of the robots could be such that: robot r_A will do a symmetric move to a point in P, robot r_{AB} a symmetric move to P_B (see the red path in Figure 10), and so on. Let t be the first round at which r moves to a location in $\{p_A, p_B, p_{AB}\}$. Since $\mathcal{A}$ solves Visit-All such a round t must exist, and w.l.o.g let us suppose that r visits first the location p_{AB}. By symmetry considerations we have that, at round t, also robot r_{AB} has to move to location p (recall that p is the location occupied by r in C_0). Therefore, from round 0 to t robots r and r_{AB} have walked on a closed path, let it be CP, between some locations of C_0 that includes p and p_{AB} but which does not include p_A, p_B (this comes directly from the definition of t). The CP path is represented in Figure 10 as the union of the red path, the blue paths and the clouds; clouds represent the move made by r and r_{AB} between rounds 3 and t. Since $\mathcal{A}$ is an algorithm for oblivious robot, this implies that robots r and r_{AB} will only visit points in CP also in future rounds. Therefore, $\mathcal{A}$ cannot be correct.

□

To summarize, we have the following:

Theorem 11. *In systems without chirality,* Visit-All *is solvable in* 1*-step if and only if* $C_0 \notin \mathcal{C}_\odot$ *and either there are no symmetry axes in* C_0*, or there exists a unique symmetry axis that does not intersect any point of* C_0*.*

4.3 2-step Algorithms

In this section we show that using 2-step algorithms does not help in enlarging the class of solvable configurations.

4.3.1 Move-All

Obviously, Theorem 4 holds also when there is no chirality. One may wonder whether is possible to solve Move-All with a 2-step algorithm when C_0 has a symmetry axis with a unique robot on it. However, the argument used in Theorem 4 can be adapted also for this case showing that this is impossible.

Theorem 12. *When there exists an axis of symmetry in* C_0 *containing a single robot and there is no chirality, then there exists no* 2*-steps algorithm that solves* Move-All*.*

Proof. Consider a configuration C_0 and let A be an axis with a single robot on it. Let us assume that algorithm $\mathcal{A}$ is a correct 2-step algorithm that solves Move-All starting from C_0, and let $\mathcal{R}_{\mathcal{A},C_0} : (C_0, C_1, C_2, C_3, \ldots)$. Let r be the robot in A and let p be its position. In configuration C_1 robot r has to move to break the symmetry in such a way that in configuration C_2 another robot could substitute r on A. Suppose the contrary, if r remains on the axis in C_1 the all the other robots will be symmetric. Therefore, in C_2 is not possible for a single robot, different than r, to reach position p. Therefore let p' be the position of r in C_1.

Let r' be the robot that exchanges position with r in configuration C_2. It is clear that if r' has a different local reference system than r, then in configuration C_3 it will move to a position $p'' \neq p'$. Therefore, we have $C_3 \notin \Pi(C_1)$, violating the Move-All specification (see Lemma 1). □

From the previous theorem and Theorem 4, we have:

Theorem 13. *When the system has no chirality, there exists no 2-step algorithm that solves* Move-All *from an initial configuration $C_0 \in \mathcal{C}_\odot$ nor when there exists a unique symmetry axis in C_0 containing a single robot.*

4.3.2 Visit-All

Theorem 14. *When $n > 2$ and there is no chirality,* Visit-All *is not solvable in 2-steps, from an initial configuration C_0, if one of the following holds:*

- *$C_0 \in \mathcal{C}_\odot$*
- *There exists a symmetry axis A of C_0 intersecting a proper non-empty subset of C_0.*
- *There are at least two symmetry axes of C_0.*

Proof. We prove the theorem case by case:

1. $C_0 \in \mathcal{C}_\odot$: this case derives directly from Theorem 4.
2. There exists a symmetry axis in C_0 and it intersects a proper non-empty subset of C_0.
   Let us assume that algorithm $\mathcal{A}$ is a correct 2-step algorithm that solves Visit-All starting from C_0, and let $\mathcal{R}_{\mathcal{A},C_0} : (C_0, C_1, C_2, C_3, \ldots)$. It is clear that, to break the symmetry, in each configuration C_j, with j odd, the symmetry has to be broken by moving a robot on an axis. Let p be the position of this robot in C_{j-1}, note that such a position is the always the same: the decision on which robot has to move is taken always on configurations that are permutations of C_0, thus it will always move a robot to the same specific position. Let r_x and r_y be two robots with different local references systems. Since $\mathcal{A}$ solves Visit-All there must exist a configuration C_{j_x} where robot r_x is in position p; analogously, there must exist a C_{j_y} where robot r_y is in position p. In configuration C_{j_x+1} robot r_x has to move outside of the axis. The same happens to r_y in configuration C_{j_y+1}. However, the reference systems could be such that $C_{j_x+1} \notin \Pi(C_{j_y+1})$, and by Lemma 1, this violate the specification of Visit-All: r_x and r_y decides where to move by looking at exactly the same snapshot; therefore, they will move on opposite locations of the axis if their local reference systems have opposite chirality.
3. There are at least two symmetry axes of C_0: first of all we can assume that it does not exist an axis of C_0 containing robots, otherwise we boil down to previous case. For any possible movements of robots, they have to keep the same symmetry. This implies that the argument of Thm. 10 still applies.

□

5 Oblivious Robots with Visible Axes and Chirality

In this section, we assume that each robot can see the axes of all robots, and that there exists a common chirality. As we have seen in Section 3, the only configurations in which Visit-All cannot be solved are the ones in $\mathcal{C}_\odot$. In Algorithm 5 we present a Voting algorithm that solves Visit-All also

starting from these configurations, provided that robots' axes are visible. The algorithm uses Procedure innerPolygon, that takes a configuration C and returns only the points on the smallest non degenerate circle with center in the centroid of C (as reference see the white points in Figure 11a).

When $C_0 \notin \mathcal{C}_\odot$, the algorithm uses the Order procedure of Algorithm 2 in Section 3. In case the initial configuration is in $\mathcal{C}_\odot$, the robots use the visible axes to implement a voting procedure. Such a procedure elects a unique vertex among the ones of the innermost non-degenerate regular polygon (see, an example in Figure 11). The vote of a robot r is computed by translating its reference system to the center of configuration C_0. The vote of r will be given to the point of the innermost non-degenerate polygon that forms the smallest counter-clockwise angle with the x axis of the translated system. Once the voting is completed, the elected point is used to break the symmetry and to compute a total order among the robots. When the total order is computed, the robots move cyclically solving Visit-All.

Algorithm 5 Order Algorithm when robots have visible axes.

procedure getVote(Polygon P, robot r)
  $o = getCenter(P)$
  Le robot p_v in P be the robot that forms the smallest clockwise angle with the x-axis of the reference system Z_r of robot r, when Z_r is translated in o.
  return p_v
procedure Voting(Configuration C)
  $P = innerPolygon(C)$
  V=vector of size $|P|$ with all entries equal to 0.
  for all $r \in C$ **do**
    $r_v = getVote(P, r)$
    $V[v] = V[v] + 1$
  p_l = elect one robot in P using the votes in V.
  return p_l
procedure Order(Configuration C)
  if $C \in \mathcal{C}_\odot$ **then**
    $p_l = Voting(C)$
    Compute a cyclic order on positions in C using the leader robot p_l.
  else
    Compute an order using Order(C) of Algorithm 2 in Section 3.1.
  return the cyclic order computed

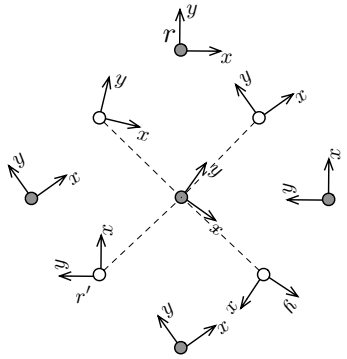
Theorem 15. *If each robot can see the axes of the others and there is chirality, then there exists a* 1*-step algorithm solving* Visit-All *for any initial configuration* C_0*.*

Proof. We have to prove the correctness of Algorithm 5 only for configurations in $\mathcal{C}_\odot$. Let k be the symmetricity of the initial configuration C_0 without the central robot, and let P be the innermost non degenerate $\beta \cdot k$-gon in C_0 with $\beta \in \mathbb{N}^+$.

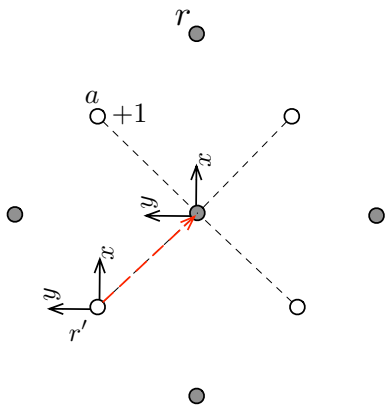
The key observation is that $n = (\alpha + \beta) \cdot k + 1$ for some $\alpha \in \mathbb{N}^+$. We now show that no matter which robot calls procedure Voting, the procedure returns always the same point in P. Voting iterates over all robots, and computes the vote of each robot $r \in C_0$. The vote is computed by first translating the reference system of r to the center of configuration C_0 and then taking as voted robot r_v, the one that makes the smallest counter-clockwise angle with the x-axis of the translated reference system. An example of the voting procedure is in Figure 11. First of all note that the result of the Voting procedure is independent from the robot that is executing it, and it always returns the same distribution of votes for points in P, even if we permute the robots in C_0.

We now show that the distribution of votes cannot be symmetric, and one robot in P can be elected.

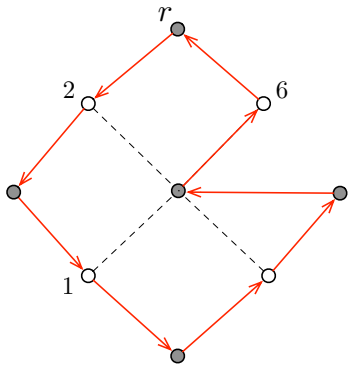
The proof is by contradiction. Let us assume that there exists an axial symmetry on the distribution of votes. The symmetry axis may cross two robots, one robot, or none. In case it crosses only one robot, then we elect that robot as leader. If the axis crosses two robots or none, then the number of votes must



(a) Initial configuration. The robots on the vertices of the innermost polygon P are the white ones.



(b) Robot r computes the vote of r': it first translate the coordinate system of r' in the central robot, then it assigns the vote to vertex a in P. This vertices is voted since is the one, among all the other vertices of P, that forms the smallest counterclockwise angle with the axis x of r'.



(c) Votes distribution and induced cyclic order.

Figure 11: Robots with visible axis, voting procedure.

be even and so the number of points in P, that is $\beta \cdot k$; but this is obviously impossible since $\beta \cdot k$ and n are co-prime.

Let us assume that there exists a rotational symmetry on the votes. Then, there exists a proper divisor $d > 1$ of $\beta \cdot k$ and an ordering of the robots in P such that $d \cdot (V[1] + V[2] + \ldots + V[\frac{\beta k}{d}]) = n$, where $V[j]$ is the number of votes for a point of P in position j of the aforementioned ordering. Notice that this would imply that n and βk are not co-prime, which is a contradiction.

It follows that it is always possible to elect a leader in P using the votes.

Therefore, procedure VOTING returns the same leader robot $p \in P$ for any permutation of the robots in C_0. It is obvious that the presence of p breaks the symmetry of the configuration and it allows to compute a total order among positions in C_0 shared by all robots. Once this total order is given the solution is immediate. □

6 Robots with Constant Memory and Chirality

Motivated by the impossibility result of Theorem 4, we investigate robots with one bit of persistent memory. Interestingly, we show that a single bit of memory is sufficient to overcome the impossibility, and solve VISIT-ALL using a 2-step algorithm. Note that we cannot overcome the impossibility using 1-step algorithms, as Theorem 2 holds even if the robots are equipped with an infinite amount of memory.

We present the 2-step algorithm below (Algorithm 6) for $n \geq 3$ robots.

Algorithm 6 2-step VISIT-ALL with one bit of memory.

```
1: procedure INIT
2:     b = 0
3:
4: C ← LOOK
5:
6: procedure COMPUTE(Configuration C)
7:     if C ∉ C_⊙ ∧ b = 0 then
8:         Compute an order using Algorithm 2 with input C.
9:         Permute robots according to the computed order.
10:    else if C ∈ C_⊙ ∧ b = 0 then
11:        b=1
12:        if I am the central robot then
13:            Compute a destination point v =COMPUTEMOVEMENTCENTRAL(C).
14:            set destination as v
15:    else if C ∈ C_⊙ ∧ b = 1 then
16:        compute a destination point v =COMPUTEMOVEMENTNOTCENTRAL(C).
17:        set destination as v
18:    else if C ∉ C_⊙ ∧ b = 1 then
19:        (C', p', Leader) =RECONSTRUCT(C)
20:        Compute a cyclic order p_0, p_1, ..., p_{n-1} of positions in C' using the pivot robot p'.
21:        if I am the Leader then
22:            b=1
23:        else
24:            b=0
25:        if my position in C' was p_i then
26:            set destination as p_{(i+1) mod n}
27:
28: MOVE: to destination
```

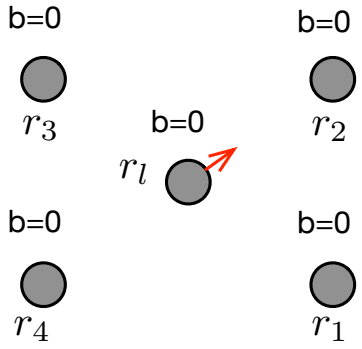
Intuitive description of the algorithm. The general idea is to use the rounds alternatingly: one round for communication, the next round to perform the actual permutation. In the communication round, the robots will be able to reconstruct the initial configuration; in the subsequent round they will be able to move.

The bit is crucial for the robots to understand their role and to distinguish the situations. Initially every robot has the bit b set to 0. Once a robot wakes up, it decides which movement to make based on b and on the current configuration C. First of all, if the initial configuration $C_0 \notin \mathcal{C}_\odot$ then the robots follow the algorithm described in Section 3, which, as we will see, does not conflict with the one for $C_0 \in \mathcal{C}_\odot$ described below.

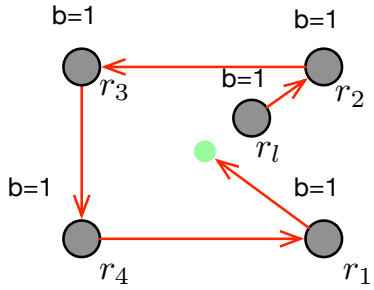
When a robot wakes up and it sees that the configuration is in $\mathcal{C}_\odot$ and b is 0, it sets the bit to 1 to remember that the algorithm is starting from a central configuration. Moreover, the central robot r_l takes the role of *Leader* and does a special move so to create a new configuration C_1 that is not in $\mathcal{C}_\odot$ but from which it is easy to reconstruct configuration C_0 (this is done by using procedure ComputeMovementCentral, whose implementation details are explained in the next paragraph). A key point of the algorithm is to keep the role of the *Leader* robot invariant.

At the next activation, the robots wake up in a configuration that is not in $\mathcal{C}_\odot$ but $b = 1$; they use the bit information to understand that they have to move in such a way to permute and reconstruct a central configuration C_2. With the exception of the Leader r_l that keeps its bit set to 1, all the other robots will reset their bit b to 0.

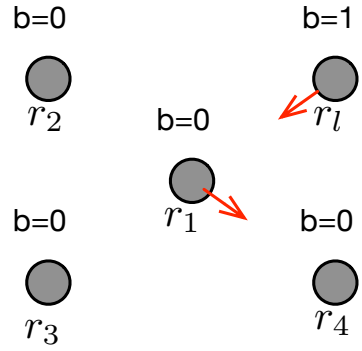
In the next configuration C_2, the central robot r_c is not r_l, because the robots have done one cyclic permutation. At this point the robots have their bit $b = 0$, with the exception of r_l who is the unique robot to see its bit $b = 1$ in a central configuration. In C_2 the central robot moves (like the central robot moved in C_0). However, also the leader robot r_l moves, and it does so in a specific way so to allow, at the next round, the reconstruction of the initial configuration (see procedure ComputeMovementNotCentral, whose implementation details are explained in the next paragraph). The combination of moves of the leader and the central robot allows the robots to reconstruct the initial configuration and to uniquely identify the leader (the reconstruction is executed by procedure Reconstruct whose details are discussed below). The cyclic order used for the permutation is computed using a "pivot" position, indicated by the leader, that occupies a special invariant point in the plane (see Figure 14). The fact that the leader is unique guarantees that the robots agree on the same permutation.
A pictorial representation of the algorithm is presented in Figure 12.



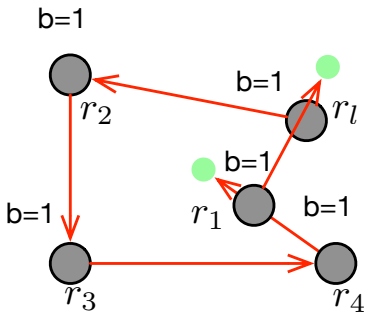
(a) round $r = 0$ (*communication round*): only the central agent r_l moves.



(b) round $r = 1$ (*permutation round*). The arrows show the cyclic order over the positions of the original central configuration.



(c) round $r = 2$ (*communication round*). r_l moves since it has $b = 1$ and r_1 moves because it is a central robot.



(d) round $r = 3$ (*permutation round*). The robots move according to the cyclic order on the reconstructed central configuration.

Figure 12: Case of central robot moving when $n > 3$.

Movements of r_l and r_c. This paragraph discusses the implementation details of functions Com-

puteMovementCentral and ComputeMovementNotCentral used in Algorithm 6. In particular, special care has to be taken to design the movements of the robot leader r_l and of the central robot r_c, if different from r_l. Such movements have to be done in such a way to break the symmetry of the configuration by electing always the same pivot position p', and to make the central configuration reconstructable. Let C be a generic central configuration in $\Pi(C_0)$. Let $P_0, P_1, \ldots, P_m$ be a decomposition of C in concentric circles, where each P_j is a circle, P_0 is the degenerate circle constituted by the only central robot, and P_1 is the innermost non-degenerate polygon on which p' resides (see Figure 13).

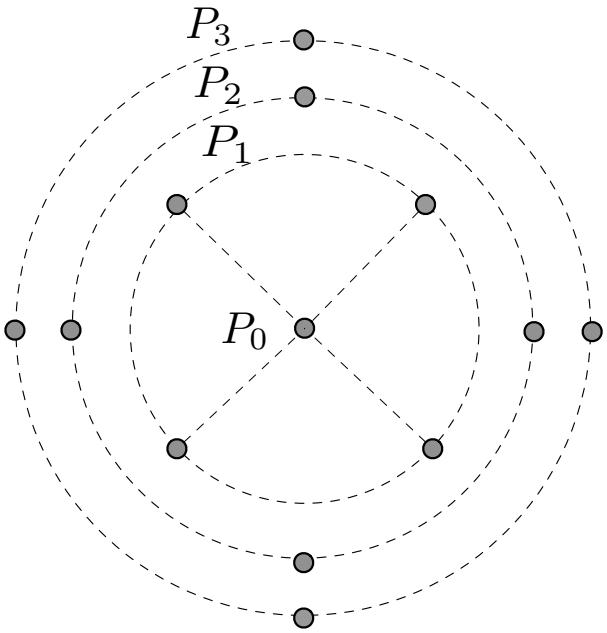


Figure 13: Decomposition of a central configuration in concentric circles. Note that P_0 is the degenerate circle formed by only the central robot.

The function ComputeMovementCentral, called by robot r_c, executes the following movements according to the number of robots n:

- $n = 3$ (C1). In such a case, since $C \in C_\odot$, the robots are on a single line. Let s be the segment containing all robots. Robot r_c moves perpendicularly to s of a distance $d = \frac{s}{2}$. The direction is chosen such that the pivot position p' will be the one of the first robot encountered travelling along the arcs that connect the endpoints of s and r_c in the clockwise direction (see Figure 16a).
- $n > 3$ (C2). Robot r_c chooses a robot position p' on P_1 as pivot. It moves on the segment connecting c and p' of a small distance (much smaller then the radius of P_1). See Figure 14.

Robot r_l, when different from r_c, executes the following movements according to n (such movements are computed by the function ComputeMovementNotCentral):

- $n = 3$. (L1) (Robots on a single line). Robot r_l moves on s by a small distance, creating a new segment s'. The pivot position p' will be indicated by the direction that goes from the old center of s to the new center of s' (see Figure 16b).
- $n > 3$. We have three sub-cases depending on which P_j robot r_l is positioned, and on the number of other robots on P_j. Note that $r_l \notin P_0$ (otherwise it would be central). Remember that P_m is the

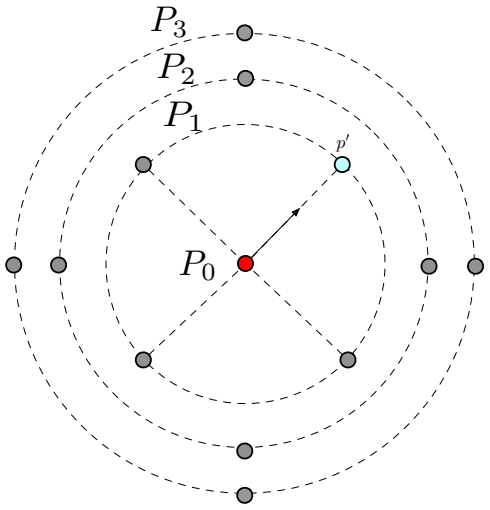


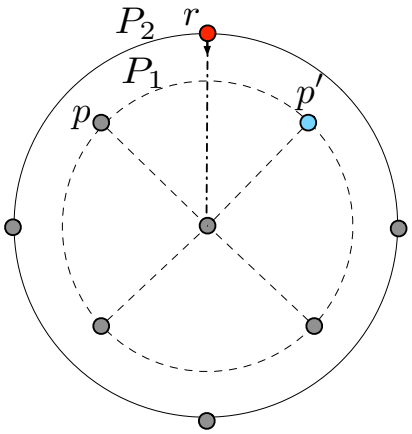
Figure 14: Movement of the central robot r_c: it moves towards position p' of a distance that is much smaller than the radius of P_1. The pivot point p' is on P_1

outermost circle, and it coincides with the SEC of C. We treat each P_j as a set, e.g. $|P_j|$ indicates the number of robots/positions in P_j. Let x be the difference between the radii of P_{j-1} and P_j, and let h be the segment connecting P_0 and robot r_l. Let p be the position of the first robot on P_1 encountered by walking in clockwise direction starting from the intersection between h and P_1. Let *nhop* be the number of positions between p and the robot p' that r_l wants to indicate. Recall that p' is the robot that r_l will move towards if r_l was in P_0.

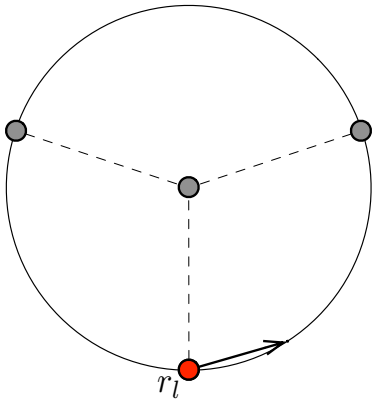
- Sub-case (L2.1). When $P_j \neq P_m$ or $P_j = P_m$ and $|P_m| > 3$: Robot r_l moves on h towards P_j by a quantity $encode(nhop) * \frac{x}{2}$. Where *encode* is an appropriate function from $\mathbb{N}$ to $(\frac{1}{2}, 1)$. See Figure 15a for an example.
- Sub-case (L2.2). When $P_j = P_m$ and $|P_m| = 3$: Note that P_m has to be rotationally symmetric; therefore it contains 3 robots each of them forming an angle of $\frac{2\pi}{3}$ with its adjacent neighbours. Robot r_l moves to a point of P_j that creates with its counter-clockwise neighbour an angle that is $encode(nhop) * \frac{2\pi}{3}$, where *encode* is an appropriate function from $\mathbb{N}$ to $(\frac{1}{2}, 1)$. See Figure 15b for an example.
- Sub-case (L2.3). When $P_j = P_m$ and $|P_m| = 2$: let s be the segment connecting the two robots on P_j. Robot r_l moves on s, expanding P_j in such a way that the new diameter is $2D + encode(nhop) * D.W$, where *encode* is an appropriate function from $\mathbb{N}$ to $(\frac{1}{2}, 1)$. See Figure 15c for an example.

Note that is not possible to have $|P_m| = 1$, since C is rotationally symmetric.

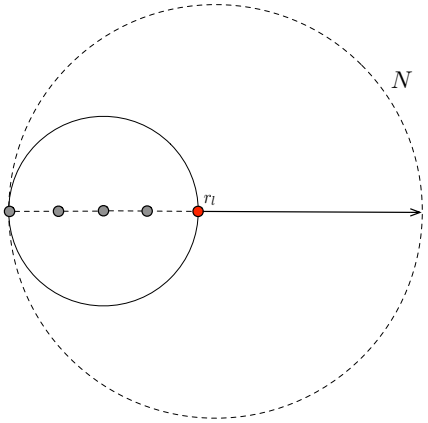
It is easy to see that when r_c (or r_c and r_l) move, the resulting configuration C' is not in $\mathcal{C}_\odot$.



(a) L2.1. The non-central leader robot is not on the SEC, or it is on the SEC with at least other three robots.



(b) L2.2. The non-central leader robot is on the SEC, and the SEC contains exactly three robots.



(c) L2.3. The non-central leader robot is on the SEC, and the SEC contains exactly two robots.

Figure 15: Movements of non-central leader robot r_l when $n \geq 4$.

Reconstruction of the central configuration. When the memory bit is 1 and the configuration C' is not in $\mathcal{C}_\odot$, the robots know that they have to (1) reconstruct the original configuration C, (2) find the pivot position p' signalled by the leader, and (3) identify the leader robot r_l. The reconstruction is executed by the function RECONSTRUCT, and is different depending on the number of robots:

- In case $n = 3$: the robots are forming a triangle. The base of the triangle is its largest edge e. The algorithm uses the height of the triangle, w.r.t. the base e to understand if r_l was r_c or not (see Figure 16c).
  - If the height of the triangle is exactly half of e, the algorithm infers that $r_l = r_c$ and that the two other robots are the endpoints of e, case (C1). The pivot p' is the position of the first clockwise robots encountered following the arc that includes robot r_l and the other two. The original configuration C is easily reconstructed: the endpoints of e are in the same position, and the central robot will be in the intersection of the perpendicular segment that goes through r_l and e.

– If the height is slightly less, or slightly more, than the largest edge e. Then the algorithm infers that r_l was one of the endpoint; we are in case (L1.1). The reconstruction of C is simple: take the intersection x of the perpendicular segment that goes through r_l and e, the position of the endpoints in C is reconstructed using the fact that the hight of the triangle is exactly half of the original segment, and that x was the center of the original segment. Robot r_l is the endpoint that moved, and the position p' is decided evaluating if r_l moved towards or away from the old center.

- In case $n > 3$: The algorithm starts by examining P_m, in order to understand if r_l was on the SEC and executed the sub-case (L2.2) or (L2.3). If the test is negative it proceeds using an ‘‘onion peeling" approach, in which the algorithm, starting from the outermost P_m, progressively examines each P_j until it finds an asymmetry or it reaches P_0. The onion peeling proceeds by first computing the SEC, that is P_m, and then computing each P_j by finding progressively smaller concentric circles.

  – Test for case (L2.3): This test case is done only on P_m. If the center of P_m is not contained in $\mathrm{SEC}(C' \setminus P_{m-1})$, then the algorithm detects case (L2.3). P_m is adjusted to a new one that has the diameter equal to the distance between the two furthest robots on P_m. Robot r_l will be the robot on P_m that is farthest from robots in P_{m-1}. The reconstruction of the last layer is done by knowing that it will be a circle with the same center of $\mathrm{SEC}(C' \setminus P_{m-1})$ that passes through $P_m \setminus \{r_l\}$ and finally p' will be indicated by decoding the information encoded in the diameter of P_m.

  – Test for case (L2.2): This test case is done only on P_m. If $|P_m| = 3$ and it is not rotationally symmetric, and ($|P_{m-1}| > 1$ or $m - 1 = 0$), then the algorithm detects case (L2.2). Robot r_l is one that is not forming an angle of $\frac{2\pi}{3}$ radians with any of it adjacent neighbours, position p' is encoded in the smallest angle that r_l is forming. The original position of r_l can be easily reconstructed: it is the one that forms an angle of $\frac{2\pi}{3}$ with each of its adjacent robots.

  – Test for case (L2.1): This test case is done on layers different than P_m. If $|P_j| = 1$ then the algorithm detects case (L2.1). Robot r_l is the only robot in P_j the reconstruction and the detection of p' are trivial.

  If the algorithm reaches P_0 not finding an asymmetry then we have that $r_l = r_c$, case (C2). The decoding is trivial: the original position of r_l is the center of P_1, position p' is the one it moved towards. Note that the algorithm uses always robot r_l to get robot p', in case $r_l \neq r_c$ also r_c moves. However, the original position of r_c is always trivial to reconstruct (the center of any P_j with $j > 0$).

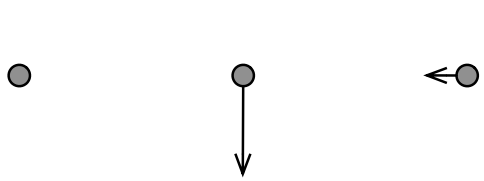
Theorem 16. *There exists an universal algorithm to solve* Visit-All *for robots with* 1 *bit of persistent memory.*

Proof. We analyse the correctness of the proposed solution considering three cases:

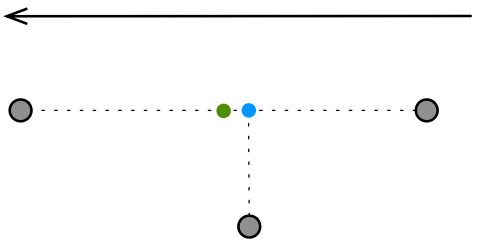
- **When the robots start in a configuration** $C \notin \mathcal{C}_\odot$: Note, that in this case all robots have bit b set to 0. This means, that the condition dictated by the `If` condition at line 7 is satisfied, thus the robots will fully execute Algorithm 2. Therefore, Visit-All will be correctly solved.

- **When the robots start in a configuration** $C \in \mathcal{C}_\odot$ **and** $n > 3$. In this case, for any central configuration reached we may alternate between two different scenario: (S1) only the central robot moves; (S2) two robots move. Scenario (S1) happens in the first configuration, and each time robot r_l goes back to the central position. Scenario (S2) happens each time we are in a central configuration, but r_l is not the central robot. We now prove the correctness for each of the aforementioned possibilities, and this is done by showing that robots start from a configuration $C \in \Pi(C_0)$, they reach a configuration $C' \notin \mathcal{C}_\odot$, and then again a configuration $C'' \in \Pi(C_0)$ that is a cyclic permutation of C_0.

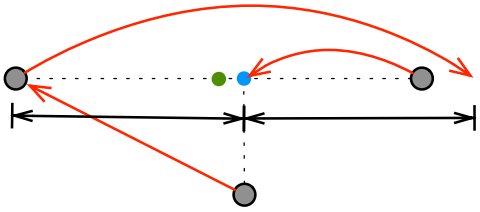
  – **Scenario (S1), only the central robot moves.** At the beginning, the robots have bit $b = 0$ and the configuration is $\mathcal{C}_\odot$, this implies that only the central robot r_l move, see line 12 of Algorithm 6, all the robots set their bit b. The central robot r_l moves of a small distance d



(a) The case of 3 robots, the central robot moves perpendicularly to initial line, and it will indicate the left robot as direction for the permutation.



(b) The external robot moved and it fixed as direction for permutation the one that goes from the old center, blue dot, to the new one, green dot.



(c) The exact positions of robots are easily reconstructable: the new central robot has to be places in the blue dot, and the left central robot has to move to its simmetric position with respect to the old center, the blue dot.

Figure 16: Special case $n = 3$ robots.

towards a robot p'. At the next round, the robots will se a $C' \notin \mathcal{C}_{\odot}$, but since $b = 1$ they know that they started from a central configuration, see line 18. Notice that, if r_l moved of a small distance still being inside the circle formed by the innermost robots (see as reference Figure 12) then it is simple for all robots to reconstruct the proper C by seeing C' and executing the RECONSTRUCT procedure. Therefore, they can understand that r_l is the only robot that moved and everyone appoints r_l as *leader*, it also simple to identify the pivot robot p' in the innermost circle of P_1.

By using this direction, the robots compute a common cyclic order on the positions in C'' (see Figure 12) and everyone moves according to this order and its position in C'', reaching a permutation of $C'' \in \Pi(C)$, before moving, all robots but the appointed leader update their bit b (line 24).

– **Scenario (S2), only two robots move.** Robots that change position are the central robot r_c (line 12 of Figure 6), and the only one that started the round with bit $b = 1$ that is robot r_l (line 15 of Figure 6). Before ending the round, all the robots set their bit b (line 11). The movement of robot r_l is different depending on its position. The purpose of this move is to indicate the pivot robot p', that r_l would have chosen if it was the central robot. After r_l and c moved, we are in a configuration C' and all robot have bit b set, so they can remember they were starting from a central configuration. The only item left to show is that they can all reconstruct the central configuration I and they can all agree on robot r_l being the leader. We now show that the robots do reconstruct C starting from C' and thus agree on the same cyclic order. The reconstruction is done by the procedure RECONSTRUCT. We have to show that for each possible case the reconstruction is correct.

  * Reconstruction from a configuration created with move (L2.1). First of all the algorithm cannot mistake it for cases (L2.2) and (L2.3). The correct reconstruction is immediate.
  * Reconstruction from a configuration created with move (L2.2). The case cannot be mistaken for (L2.3): the SEC is concentric with the other P_j. It cannot be mistaken for case L2.1: $|P_{m-1}| > 1$ or $m - 1 = 0$. The reconstruction is immediate.

* Reconstruction from a configuration created with move (L2.3). This case is detected by observing that the centre of SEC is not contained in $\mathrm{SEC}(C' \setminus P_m)$. This happens only when r_l moved according to rule (L2.3). The RECONSTRUCT adjusts the SEC, that contains three robots, to a new sec P_m that has as diameter the two furthest robots on the old SEC. Let r_l and r_x be the two robots on the new P_m. We argue that the distance of this two robots is encoding the position p'. Refer to Figure 15c: robot r_l moves from the other robots up to a distance $2D + x * D$. By doing so, it is becoming the robot that is furthest from all the others in C'. It is also clear that the by removing r_l and r_x we get $C \setminus \{r_l, r_x\}$ (neglecting for now the central robot). The diameter D can be recovered by taking the circle with center in $\mathrm{SEC}(C \setminus \{r_l, r_x\})$ and passing through r_x. With this information the decoding is simple, and the reconstruction of C is immediate.

– **When the robots start in a configuration $C \in \mathcal{C}_\odot$ and $n = 3$**: In this case when the configuration is in $\mathcal{C}_\odot$ the movements of the central robot and of the leader robot r_l follow special rules. Notice that all three robots lie on a segment s. Robot r_l moves following case (L2.1) if is not central. The central robot moves according to case (C2). It easy to see that the robots reconstruct the central configuration and they correctly elect the leader, which is either the central robot (if it is the only robot that moved) or the external robot (if two robots moved). It is also immediate that VISIT-ALL is solved, since robots execute continuously the same cyclic permutation.

Once C, p', r_l are known the robots agree on the same cyclic order and reach a new configuration $C \in \Pi(C_0)$. An immediate induction shows that starting from the initial configuration C_0 we alternate either in scenario S1 or S2 solving VISIT-ALL.

□

7 Concluding Remarks

To the best of our knowledge, this is the first investigation of the problems of permuting the positions of a set of oblivious mobile robots. Surprisingly this class of problems seems to be more difficult than the previously studied problems such as gathering and pattern formation, which have easy solutions for the strongest model of fully synchronous robots with rigid movements. Thus the characterization of solvable instances for permutation problems is quite different as shown in this paper. Moreover we also showed that being non-oblivious is helpful for permuting robots, unlike the formation problems where the solvability is unaffected by obliviousness [20].

The paper opens several research directions that are worth investigating: an interesting direction would be to discover other class of problems which cannot be solved even when it is easy to elect a leader (as the class of problems considered here). The difficulty in solving the permutation problems seems to be unrelated to agreement problems such as leader election. In particular we may try to study the differences between leader election and permutation problems and determine if the latter is strictly more difficult than the former. We may also consider other interesting assumptions that can help in overcoming the challenges for permuting robots without orientation. An example could be to investigate if the presence of visible lights [5, 7, 10, 17] could help. Our result of Section 6 shows that internal lights are enough in case of chirality, without chirality the question remains open.

Acknowledgement: This work has been partially funded by the University of Rome "La Sapienza" with the Calypso project.

References

[1] H. Ando, I. Suzuki, and M. Yamashita. Formation and agreement problems for synchronous mobile robots with limited visibility. In *Proc. of the 10th IEEE Symp. on Intelligent Control*, pages 453–460, 1995.

[2] Q. Bramas and S. Tixeuil. Brief announcement: Probabilistic asynchronous arbitrary pattern formation. In *Proc. of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 443–445, 2016.

[3] S. Cicerone, G. Di Stefano, and A. Navarra. Asynchronous embedded pattern formation without orientation. In *Proc. of the 30th International Symposium on Distributed Computing (DISC)*, pages 85–98, 2016.

[4] S. Cicerone, G. Di Stefano, and A. Navarra. Gathering of robots on meeting-points: feasibility and optimal resolution algorithms. *Distributed Computing* 31(1): 1–50, 2018.

[5] S. Das, P. Flocchini, G. Prencipe, N. Santoro, and M. Yamashita. Autonomous mobile robots with lights. *Theoretical Computer Science*, 609(P1):171–184, January 2016.

[6] S. Das, P. Flocchini, N. Santoro, and M. Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. *Distributed Computing*, 28(2):131–145, April 2015.

[7] G.A. Di Luna, P. Flocchini, S. Gan Chaudhuri, F. Poloni, N. Santoro, G. Viglietta, Mutual visibility by luminous robots without collisions, In *Information and Computation* 254(3) 392–418, 2017.

[8] G. Di Luna, P. Flocchini, N. Santoro, G. Viglietta. TuringMobile: A turing machine of oblivious mobile robots with limited visibility and its applications. In *Proc. of the 32nd International Symposium on Distributed Computing (DISC)*, pages 19:1-19:15, 2018.

[9] G. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and M. Yamashita. Meeting in a polygon by anonymous oblivious robots. In *Proc. of the 31st International Symposium on Distributed Computing (DISC)*, pages 14:1-14:15, 2017.

[10] G. Di Luna, G. Viglietta. Robots with lights. In *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 252-277, 2019.

[11] M. Erdelj, T. Razafindralambo, and D. Simplot-Ryl. Covering points of interest with mobile sensors. *IEEE Transactions on Parallel and Distributed Systems* 24(1):32–43, 2013.

[12] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Morgan & Claypool, 2012.

[13] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous oblivious robots. *Theoretical Computer Science*, 407(1-3):412–447, 2008.

[14] P. Flocchini, N. Santoro, G. Viglietta, and M. Yamashita. Rendezvous with constant memory. *Theoretical Computer Science*, 621:57–72, 2016.

[15] N. Fujinaga, Y. Yamauchi, H. Ono, S. Kijima, and M. Yamashita. Pattern formation by oblivious asynchronous mobile robots. *SIAM Journal on Computing*, 44(3):740–785, 2015.

[16] N. Fujinaga, Y. Yamauchi, S. Kijima, and M. Yamashita. Asynchronous pattern formation by anonymous oblivious mobile robots. In *Proc. of the 26th International Symposium on Distributed Computing (DISC)*, pages 312–325, 2012.

[17] G. Sharma, C. Busch, and S. Mukhopadhyay. Mutual visibility with an optimal number of colors. In *11th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (Algosensors)*, pages 196–201, 2015.

[18] K. Sugihara and I. Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *Journal of Robotics Systems*, 13:127–139, 1996.

[19] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[20] M. Yamashita and I. Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theoretical Computer Science*, 411(26-28):2433–2453, 2010.

[21] Y. Yamauchi and M. Yamashita. Randomized pattern formation algorithm for asynchronous oblivious mobile robots. In *Proc. of the 28th International Symposium on Distributed Computing (DISC)*, pages 137–151, 2014.