# Oblivious Permutations on the Plane

**Shantanu Das**
Aix-Marseille University and CNRS, LIS, Marseille, France
shantanu.das@lis-lab.fr

**Giuseppe A. Di Luna**[1]
DIAG, University of Rome “Sapienza”, Italy
diluna@diag.uniroma1.it

**Paola Flocchini**
University of Ottawa, Ottawa, Canada
paola.flocchini@uottawa.ca

**Nicola Santoro**
Carleton University, Ottawa, Canada
santoro@scs.carleton.ca

**Giovanni Viglietta**
JAIST, Nomi City, Japan
johnny@jaist.ac.jp

**Masafumi Yamashita**
Kyushu University, Fukuoka, Japan
masafumi.ymashita@gmail.com

## 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 studied formation problems, 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.

**2012 ACM Subject Classification** Theory of computation → Distributed algorithms; Computer systems organization → Robotics

**Keywords and phrases** Distributed Algorithms, Mobile Robots, Fully synchronous, Oblivious, Permutations, Chirality, Sequence of configurations

**Digital Object Identifier** 10.4230/LIPIcs.OPODIS.2019.24

**Related Version** `https://arxiv.org/abs/1911.05239`

**Funding** *Paola Flocchini*: Natural Sciences and Engineering Research Council of Canada (N.S.E.R.C.) under the Discovery Grant program.

[1] Contact Author. Address: DIAG, Via Ariosto 25, Roma, Italy.

## 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* 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. If the robots agree on a common notion of clockwise direction, then we say the system has *chirality*.

Some typical problems that have been studied in this model include: *gathering* of robots (e.g., [11, 12]), uniform *dispersal*, *filling* a region with robots, *flocking*, etc. (for a review, see [6]). 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, 5, 7, 8, 14, 15, 16, 17]). A major issue in such formation problems is the amount of symmetry (quantified by the notion of symmetricity [15]) 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, 9].

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 [5], 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 chirality and the coordinate 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 chirality, it is possible to solve the problems for all initial configurations (Section 6).

Due to the space constraint, some of the proofs and formal description of some algorithms have been omitted, they can be found in the full version.

## 2 Model, Definitions and Preliminaries

**Robots and scheduler.** We consider a set of dimensionless computational entities: the *robots*. These robots are modelled as points in the metric space $\mathbb{R}^2$; they are able to sense the environment detecting the presence of other robots, they can perform computations, and are able to move to any other point in the space. Each robot has its own local coordinate system centred in its own position (which may differ in orientation and unit distance from the coordinate 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 [6]. During the *Look* phase, a robot $r$ takes an instantaneous *snapshot* of the environment, the snapshot is an entire map of the plane containing positions of all the other robots with respect to the local coordinate system of $r$. During the *Compute* phase, robot $r$ performs some local computation to decide its new 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* [6], 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 at most $m$ where robots in the same class have the same view (snapshot) in $C$ (See [15, 16]). 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 are the only configurations that allow to elect a leader among robots). For symmetric configurations with $\rho(C) > 1$, $C$ may have rotational symmetry with respect to the center $c$ of $SEC(C)$, which coincides with the centroid of $C$ in this case, or $C$ may have mirror symmetry with respect to a line, called the axis of symmetry.

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 $SEC(C)$ such that $\rho(C \setminus \{r_c\}) = k > 1$; In other words, $C$ has a rotational symmetry around $r_c$ such that $C$ can be rotated around centre $r_c$ by an angle $\theta = \frac{\pi}{k}$ to obtain a permutation of $C$. Figure 2 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 $\pi(C) = (p_{\pi(0)}, p_{\pi(1)}, \ldots, p_{\pi(n-1)})$. We denote: (1) the set of permutations with no fixed points as $\Pi_2 = \{\pi : \pi(i) \neq i,\ \forall i : 0 \leq i \leq n-1\}$ and (2) the set of cyclic permutations of order $n$ as $\Pi_n = \{\pi : \pi^j(i) = i <=> nk = j$ for some $k \in \mathbb{N}\}$ where $\pi^j$ indicates that we apply permutation $\pi$ $j$ times. Let $\Pi(C)$ be the set of all permutations of $C$.

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 $t \in \mathbb{N}$. 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 coordinate systems, the algorithm 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 having chirality (i.e., they agree on the same clockwise orientation).

### 3.1 1-Step Algorithms

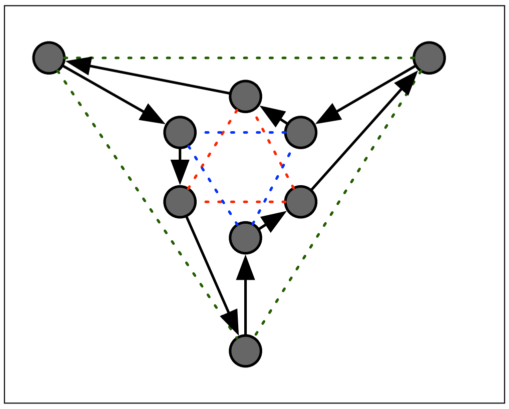
We first consider 1-step algorithms, and show that MOVE-ALL and VISIT-ALL are solvable if the initial configuration $C_0$ is not in $\mathcal{C}_{\odot}$.

**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. When the center $c$ of $SEC(C_0)$ is not occupied by 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 1 for example). The only issue is when there is a robot at $c$. 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 $p_{(i+1) \mod n}$.
```


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

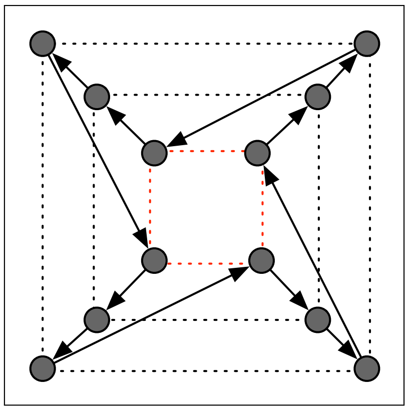

(b) Another example of cyclic order induced by the ORDER algorithm.

**Figure 1** ORDER algorithm: Examples of cyclic order computed by Algorithm 1.

▶ **Theorem 2.** *In systems with chirality, there exists a 1-step algorithm that solves* VISIT-ALL *from any initial configuration* $C_0 \notin \mathcal{C}_{\odot}$.

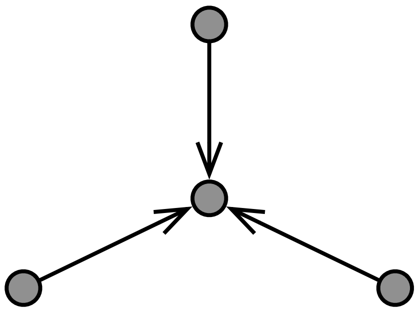


**Figure 2** Configuration $C_0 \in \mathcal{C}_\odot$ where it is impossible to solve Move-All with a 1-step algorithm.

We now show that, when $C_0 \in \mathcal{C}_\odot$ Move-All (and thus Visit-All) is unsolvable by any 1-step algorithm.

▶ **Theorem 3.** *If $C_0 \in \mathcal{C}_\odot$ there exists no* 1*-step algorithm that solves* Move-All*, even when the robots have chirality.*

**Proof.** In any configuration in $\mathcal{C}_\odot$, the adversary can assign coordinate systems in such a way that each robot, except the central robot $r_c$, has at least one analogous with a symmetric view. This derives directly 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 $C_0$, taking the position of $r_c$. An example is given in Figure 2, where if one robot moves to the centroid of $C_0$, then every robot except $r_c$ would do the same. This implies that, in the next round, it is impossible to form any $C_1 \in \Pi(C_0)$ with a different central robot. ◀

Note that Theorem 2 implies that Move-All is solvable under the same assumptions of the theorem (if we satisfy the Visit-All specification we satisfy also the Move-All specifications). Moreover, for the same reason, Theorem 3 implies that Visit-All is unsolvable. We can summarize the results of this section as follows:

▶ **Theorem 4.** *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 4, 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.

▶ **Lemma 5.** *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 $C_1 \in \mathcal{C}_\odot$.*

The above 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 $C_1 \notin \mathcal{C}_\odot$. Based on the above result, we can show the following:

▶ **Theorem 6.** *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.*

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. This shows the impossibility.

Interestingly, when we consider VISIT-ALL we can prove a stronger impossibility result that includes algorithms using any constant number of steps.

▶ **Theorem 7.** *There exists no $k$-step algorithm for* VISIT-ALL*, starting from any configuration $C_0 \in \mathcal{C}_{\odot}$, where $k = \mathcal{O}(1)$. This result holds even if the system has chirality*

## 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 8.** *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.*

We now consider the solutions to the MOVE-ALL problem for the feasible instances. If the configuration has a central symmetry (i.e., a rotational symmetry with $\theta = \pi$), each robot can be paired to its counterpart on the opposite end of the center, and the paired robots can swap positions. When the initial configuration has a rotational symmetry but no symmetry axes, then the robots can agree on a common chirality and the algorithms from the previous section can be applied. Thus the only remaining configurations are those with an axis of symmetry that are not central symmetric. For such configurations, it is possible to partition the robots in three disjoint subsets, and it make them move as follows: (see also Figure 3) (i) 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. (ii) The second subset contains robots that are closer to one symmetry axis compared to other axes. These robots swap positions pairwise, each robot switching with its symmetric robot w.r.t. the closest axis. (iii) The last subset consists of robots that are equidistant from two distinct symmetry axes. Also in this case robots switch positions pairwise, and each one switches position with its symmetric robot w.r.t. the centroid $c$ of configuration $C_0$.

For all the configurations excluded by Theorem 8, MOVE-ALL can be solved using the above approach.

▶ **Theorem 9.** *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.*

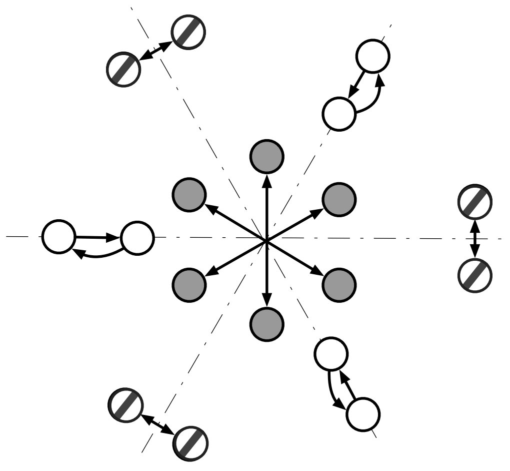


**Figure 3** Configuration $C_0$ having 3 axes of symmetry. The arrows indicate three types of robot swaps: (1) Robots on the axis agree on a cyclic order (white robots); (2) Robots that are closer to one axis of symmetry swap positions w.r.t this axis (stripped robots); (3) Robots that are equidistant from two axes, swap position with symmetric robots w.r.t. the center (black robots).

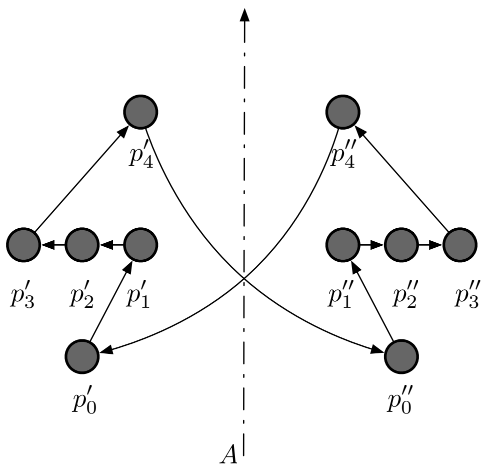


**Figure 4** Configuration $C_0$ with a unique symmetry axis $A$ and no robots on the axis. The arrow on axis $A$ indicate the direction on which robots agree. The arrows among configuration points indicate the cyclic order induced by the algorithm.

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

▶ **Theorem 10.** *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 the algorithm from the previous section 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 $[p'_0, p'_1, \ldots, p'_{\frac{n-1}{2}}]$ be the total order on $C'$ and $[p''_0, p''_1, \ldots, p''_{\frac{n-1}{2}}]$ be the analogous order on $C''$. 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 4).

▶ **Theorem 11.** *When $n > 2$ and robots do not have chirality,* Visit-All *is solvable in* 1*-step if the initial configuration $C_0 \notin \mathcal{C}_\odot$ and either*

**(i)** *There are no symmetry axes in $C_0$, or,*
**(ii)** *There exists a unique symmetry axis of $C_0$ and no point of $C_0$ intersects the axis.*

Interestingly, without chirality, Visit-All is not solvable if the assumptions of Th. 11 do not hold:

▶ **Theorem 12.** *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 $C_0$.*

To summarize, we have the following:

▶ **Theorem 13.** *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

We can show that 2-step algorithms do not help to enlarge the class of solvable configurations.

▶ **Theorem 14.** *When the system has no chirality,* Move-All *is not solvable in* 2*-steps, from an initial configuration $C_0$, if $C_0 \in \mathcal{C}_{\odot}$, or if there exists an axis of symmetry in $C_0$ containing a single robot.*

▶ **Theorem 15.** *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 $C_0$.*

## 5 Oblivious Robots with Visible Coordinate Systems

In this section, we assume that each robot can see the coordinate system of all robots and the system has chirality. As we have seen in Section 3, with chirality, the only configurations in which Visit-All cannot be solved are the ones in $\mathcal{C}_{\odot}$. We now present a Voting algorithm that solves Visit-All also starting from these configurations, provided that robots have this extra knowledge of the coordinate systems of other robots. The algorithm (see Algorithm 2) uses Procedure innerPolygon, which takes a configuration $C$ and returns only the points on the smallest non degenerate circle having the same center as $SEC(C)$ and passing through at least one point of $C$.

When $C_0 \notin \mathcal{C}_{\odot}$, the algorithm uses the Order procedure from Section 3. In case the initial configuration is in $\mathcal{C}_{\odot}$, the algorithm implements a voting procedure to elect a unique vertex of the innermost non-degenerate polygon $P$ computed by Procedure innerPolygon. The vote of a robot $r$ is computed by translating its coordinate system to the center of $SEC(C_0)$. The vote of $r$ will be given to the point of $P$ that forms the smallest counter-clockwise angle with the $x$-axis of the translated system. Since the number of robots is co-prime to the size of $P$, a unique vertex (robot) can be elected and the elected point is used to break the symmetry and compute a total order among the robots. As before, the robots use this total order to move cyclically solving Visit-All.

▶ **Theorem 16.** *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 $C_0$.*

**Algorithm 2** ORDER Algorithm when robots have visible coordinate systems.

```
procedure GETVOTE(Polygon $P$, robot $r$)
    $o = getCenter(P)$
    Consider the coordinate system with origin $o$ and axes parallel to the system $S_r$ of robot $r$. Let $p_v \in P$ be the point with smallest polar coordinates in this coordinate system.
    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 the algorithm in Section 3.1.
    return the cyclic order computed
```

## 6 Robots with one bit of Persistent Memory

Motivated by the impossibility result of Theorem 6, we now investigate non-oblivious robots having some 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 and Theorem 3, holds even if the robots are equipped with an infinite amount of memory.

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

**Algorithm 3** 2-step VISIT-ALL with one bit of memory.

```
(Initially: $b = 0$)

if $C \notin \mathcal{C}_\odot \wedge b = 0$ then
    Compute an order using Algorithm from Section 3.
    Permute robots according to the computed order.
else if $C \in \mathcal{C}_\odot \wedge b = 0$ then
    b=1
    if I am the central robot then
        Compute a destination point v =COMPUTEMOVEMENTCENTRAL($C$).
        set destination as v
else if $C \in \mathcal{C}_\odot \wedge b = 1$ then
    compute a destination point v =COMPUTEMOVEMENTNOTCENTRAL($C$).
    set destination as v
else if $C \notin \mathcal{C}_\odot \wedge b = 1$ then
    $(C', p', Leader)$ =RECONSTRUCT($C$)
    Compute a cyclic order $p_0, p_1, \ldots, p_{n-1}$ of positions in $C'$ using the pivot robot $p'$.
    if I am the Leader then
        b=1
    else
        b=0
    if my position in $C'$ was $p_i$ then
        set destination as $p_{(i+1) \mod n}$
```

**Intuitive description of the algorithm.** The general idea is to use alternate rounds of communication and formation of the actual permutation. In the communication round, the robots create a special intermediate configuration that provides a total order on the robots;

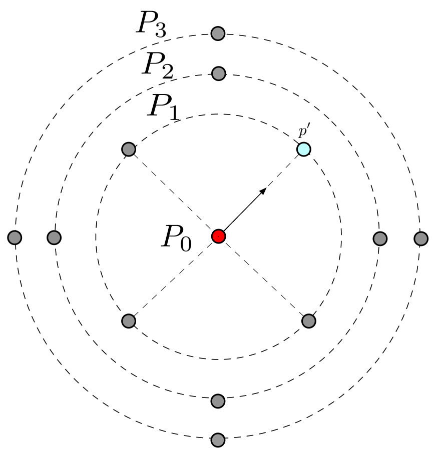


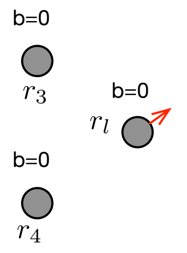
**Figure 5** Case C2: the central robot $r_c = r_l$ moves to indicate the pivot point $p'$ on circle $P_1$.

In the subsequent round they reconstruct the initial pattern forming the permutation of the initial configuration. The memory bit is crucial to distinguish the intermediate configuration from the initial configuration. If the initial configuration $C_0 \notin \mathcal{C}_\odot$ then the robots follow the 1-step algorithm described in Section 3 and we will show that this does not conflict with the rest of the algorithm designed for the case when $C_0 \in \mathcal{C}_\odot$, as described below.

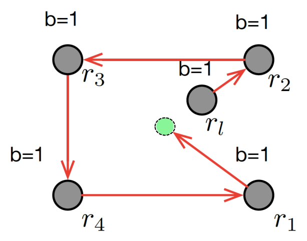
Initially every robot has the bit $b$ set to 0. When a robot observes that the configuration is in $\mathcal{C}_\odot$ and bit $b$ is 0, it sets the bit to 1 to remember that the initial configuration $C_0 \in \mathcal{C}_\odot$. The central robot $r_l$ in $C_0$ takes the role of *Leader* and performs a special move to create the intermediate configuration $C_1$ that is not in $\mathcal{C}_\odot$ but from $C_1$, it is possible to reconstruct the initial configuration $C_0$ or any permutation of it (This move is determined by procedure ComputeMovementCentral described in the next paragraph).

A key point of the algorithm is that the *Leader* robot remains invariant. At the next activation, the robots observe a configuration that is not in $\mathcal{C}_\odot$ and they have bit $b = 1$; this indicates that this is an intermediate configuration and the robots move to reconstruct a configuration $C_2 = \Pi(C_0)$. With the exception of the Leader $r_l$ whose memory bit $b$ is always set to 1, all the other robots will now reset their bit $b$ to 0.

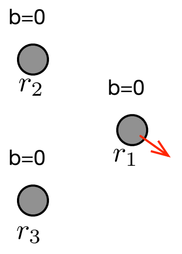
In the next round, the robots are in configuration $C_2$, where the central robot $r_c$ is not $r_l$ (the robots have performed one cyclic permutation). At this point, the robot $r_l$ is the unique robot whose bit $b = 1$. All other robots have $b = 1$ and they behave similarly as in the first round, including robot $r_c$ which moves like the central robot moved in $C_0$. However, the leader robot $r_l$ also moves at the same time, in a special way (as described in procedure ComputeMovementNotCentral presented in the next paragraph). The combination of moves of the leader robot and the central robot allows the robots not only to determine the initial configuration, but also to uniquely identify a “pivot” point in the pattern (see Figure 6), which is kept invariant during the algorithm. The recognition of the pivot point allows the robots agree on the same cyclic ordering of the points in the initial pattern, thus allowing cyclic permutations of the robots.
An example execution of this algorithm is presented in Figure 6.



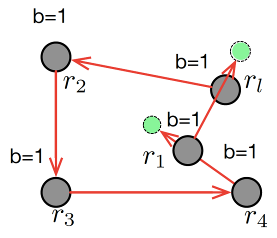
**(a)** round $r = 0$ (*communication round*): only the central agent $r_l$ moves. The pivot position is the one occupied by $r_2$.



**(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 6** Example execution of the first 4 rounds for the case $n > 3$.

**Movements of $r_l$ and $r_c$.** We now describe details of procedures ComputeMovement-Central and ComputeMovementNotCentral used in Algorithm 3. The movements of the robot leader $r_l$ and the central robot $r_c$ (if different from $r_l$) have to be designed in such a way as to break the symmetry of the configuration by electing always the same pivot position $p'$, and to make it possible to reconstruct the original configuration after the move. For any configuration $C \in \Pi(C_0)$. we define $P_0, P_1, \dots, P_m$ as a decomposition of $C$ into concentric circles, where $P_0$ is the degenerate circle consisting of only the central robot, $P_1$ is the innermost non-degenerate circle on which $p'$ is located, and finally $P_m = SEC(C)$ (see Figure 5).

Procedure ComputeMovementCentral determines the movement of robot $r_c$, according to the number of robots $n$.

- (Case C1: $n = 3$ ): In such a case, the robots are on a single line. Let $s$ be the segment of this line containing all three robots. Robot $r_c$ moves perpendicularly to $s$ of a distance $d = \frac{|s|}{2}$. The direction of movement is chosen such that the pivot position $p'$ will be the position of the first robot in the clockwise direction from $r_c$.
- (Case C2: $n > 3$): Robot $r_c$ chooses a robot position $p'$ on $P_1$ as the pivot point. Robot $r_c$ moves on the segment connecting $r_c$ and $p'$ by a very small distance (much smaller compared to the radius of $P_1$). See Figure 5.

Procedure ComputeMovementNotCentral computes the movements of the leader robot $r_l$, when different from $r_c$, again according to the value of $n$:

- (Case L1: $n = 3$ ): Note that robots are on a single line. Robot $r_l$ moves along $s$ by a small distance, changing segment $s$ to $s'$. The pivot position $p'$ will be indicated by the direction that goes from the center of $s$ to the new center of $s'$.
- (Case L2: $n > 3$ ): We have three sub-cases depending on the circle $P_j$ which contains robot $r_l$, and on the number of other robots on $P_j$. Recall that $r_l \notin P_0$. We treat each $P_j$

as a set, e.g. $|P_j|$ indicates the number of robots 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 point of intersection between $h$ and $P_1$. Let *nhop* be the number of robots in $P_1$ between $p$ and the pivot point $p'$. Recall that $|P_m| > 1$, since $P_m = SEC(C)$ and $C$ is rotationally symmetric. In the following, *encode* is an appropriately chosen function from $\mathbb{N}$ to $(\frac{1}{2}, 1)$.

- (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 $\mathit{encode}(\mathit{nhop}) * \frac{x}{2}$.
- (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 $\mathit{encode}(\mathit{nhop}) * \frac{2\pi}{3}$.
- (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 $2D + \mathit{encode}(\mathit{nhop}) * D$.

It is easy to see that after $r_c$ and $r_l$ move according to the above procedures, the resulting intermediate configuration $C'$ is not in $\mathcal{C}_{\odot}$. We now show how to reconstruct the initial configuration $C$ from $C'$.

**Reconstruction of the initial configuration.** When the current configuration $C'$ is not in $\mathcal{C}_{\odot}$ and the robots have bit $b = 1$, the robots know that they are in an intermediate configuration and they have to (1) reconstruct the original configuration $C$, (2) determine the pivot point $p'$ in $C$, and (3) identify the leader robot $r_l$. The reconstruction is performed by procedure Reconstruct, which again, depends on the value of $n$.

- If $n = 3$: the robots must form a triangle. The base of the triangle is its largest edge $e$. The algorithm computes the ratio of the height of the triangle over the length of base $e$ to determine if $r_l$ was $r_c$ or not.
  - 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 point $p'$ is the position of the first clockwise robot starting from the top of the triangle. 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 of $C$ is reconstructed using the fact that the height 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 pivot $p'$ can be computed by evaluating if $r_l$ moved towards or away from the old center.
- If $n > 3$: 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 considering 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 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 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 its 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 $P_j$ and it is trivial to reconstruct $C$ and compute the pivot $p'$.

If the algorithm reaches $P_0$ without finding any asymmetry, then we have that $r_l = r_c$ (case C2). The decoding is trivial in this case: the original position of $r_l$ is the center of $P_1$, and the pivot position $p'$ is in the direction where $r_l$ moved.

Based on the above discussion, we conclude with the following result:

▶ **Theorem 17.** *There exists an universal algorithm to solve* VISIT-ALL *for robots with* 1 *bit of persistent memory when the system has chirality.*

## 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 mobile robots in the plane. 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 [16]. 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. One possibility is the investigation of robots with the additional capability of communicating using visible lights [4, 10, 13].

## References

1 H Ando, I Suzuki, and M Yamashita. Formation and agreement problems for synchronous mobile robots with limited visibility. In *Proceedings of Tenth International Symposium on Intelligent Control*, pages 453,460. IEEE, 1995.

2 Quentin Bramas and Sébastien Tixeuil. Brief Announcement: Probabilistic Asynchronous Arbitrary Pattern Formation. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 443–445, 2016. `doi:10.1145/2933057.2933074`.

3 Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Asynchronous Embedded Pattern Formation Without Orientation. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, pages 85–98, 2016. doi:10.1007/978-3-662-53426-7_7.

4 Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. Autonomous mobile robots with lights. *Theor. Comput. Sci.*, 609:171–184, 2016. doi:10.1016/j.tcs.2015.09.018.

5 Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. *Distributed Computing*, 28(2):131–145, 2015. doi:10.1007/s00446-014-0220-9.

6 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012. doi:10.2200/S00440ED1V01Y201208DCT010.

7 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theor. Comput. Sci.*, 407(1-3):412–447, 2008. doi:10.1016/j.tcs.2008.07.026.

8 Nao Fujinaga, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita. Asynchronous Pattern Formation by Anonymous Oblivious Mobile Robots. In *Distributed Computing - 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings*, pages 312–325, 2012. doi:10.1007/978-3-642-33651-5_22.

9 Nao Fujinaga, Yukiko Yamauchi, Hirotaka Ono, Shuji Kijima, and Masafumi Yamashita. Pattern Formation by Oblivious Asynchronous Mobile Robots. *SIAM J. Comput.*, 44(3):740–785, 2015. doi:10.1137/140958682.

10 Giuseppe Antonio Di Luna, Paola Flocchini, Sruti Gan Chaudhuri, Federico Poloni, Nicola Santoro, and Giovanni Viglietta. Mutual visibility by luminous robots without collisions. *Inf. Comput.*, 254:392–418, 2017. doi:10.1016/j.ic.2016.09.005.

11 Giuseppe Antonio Di Luna, Paola Flocchini, Nicola Santoro, and Giovanni Viglietta. TuringMobile: A Turing Machine of Oblivious Mobile Robots with Limited Visibility and Its Applications. In *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, pages 19:1–19:18, 2018. doi:10.4230/LIPIcs.DISC.2018.19.

12 Giuseppe Antonio Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Masafumi Yamashita. Meeting in a Polygon by Anonymous Oblivious Robots. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 14:1–14:15, 2017. doi:10.4230/LIPIcs.DISC.2017.14.

13 Giuseppe Antonio Di Luna and Giovanni Viglietta. Robots with Lights. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, pages 252–277. Springer, 2019. doi:10.1007/978-3-030-11072-7_11.

14 Kazuo Sugihara and Ichiro Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *J. Field Robotics*, 13(3):127–139, 1996. doi:10.1002/(SICI)1097-4563(199603)13:3<127::AID-ROB1>3.0.CO;2-U.

15 Ichiro Suzuki and Masafumi Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999. doi:10.1137/S009753979628292X.

16 Masafumi Yamashita and Ichiro Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theor. Comput. Sci.*, 411(26-28):2433–2453, 2010. doi:10.1016/j.tcs.2010.01.037.

17 Yukiko Yamauchi and Masafumi Yamashita. Randomized Pattern Formation Algorithm for Asynchronous Oblivious Mobile Robots. In *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, pages 137–151, 2014. doi:10.1007/978-3-662-45174-8_10.