Weak robots performing conflicting tasks without knowing who is in their team

Subhash Bhagat
Indian Statistical Institute
Kolkata, India
subhash.bhagat.math@gmail.com

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

Krishnendu Mukhopadyaya
Indian Statistical Institute
Kolkata, India
krishnendu@isical.ac.in

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

ABSTRACT

In this paper, we consider the problem of having two teams of identical robots, each with its own task, inhabiting the same space. The robots operate in Look-Compute-Move cycles and each team needs to solve its own task without being able to distinguish which of the robots belong to its team. The tasks we consider are two classical conflicting pattern formation problems: gathering (where the robots need to gather at some arbitrary point), and circle formation (where the robots need to place themselves in distinct points of a circle). We show how to achieve this double goal using robots that are anonymous, oblivious, silent, and asynchronous; the robots share a coordinate system, but with possibly different orientations. Unlike all the previous literature, which considers a single team of robots with a single goal, this is the first result addressing multiple anonymous teams of robots performing different (and possibly conflicting) tasks in the same space at the same time.

CCS CONCEPTS

• **Theory of computation → Design and analysis of algorithms**; **Distributed algorithms**; **Self-organization**.

KEYWORDS

Swarm robots, asynchronous, oblivious, non-rigid movements, direction-only agreement, the gathering problem, the circle formation problem

ACM Reference Format:
Subhash Bhagat, Paola Flocchini, Krishnendu Mukhopadyaya, and Nicola Santoro. 2020. Weak robots performing conflicting tasks without knowing who is in their team. In *ICDCN 2020*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

1.1 Framework and Problem

The control and coordination of autonomous mobile robots have long been object of study in several fields, including robotics, control, AI, as well as distributed computing. Within distributed computing, in particular, extensive efforts have been conducted in the last two decades to investigate the computational and complexity issues arising in distributed systems composed of a team of mobile computational entities moving and operating in a Euclidean space. These entities, called *robots*, are identical in their outward appearance, homogeneous (have the same capabilities and execute the same algorithm), and without explicit means of direct communication. Each robot, when active, operates in Look-Compute-Move cycles: it determines the positions of the robots in the system (*Look*); this information is used to compute a destination point (*Compute*); the robot then moves towards the computed destination (*Move*); after the execution of a cycle, the robot may become temporarily inactive. Furthermore, the entities are oblivious: at the beginning of a cycle the robot has no recollection of computations and operations performed in previous cycles; that is, there is no persistent memory. This computational model, called $\mathcal{OBLOT}$, is the most widely investigated within distributed computing [8]. The research effort has been on determining which problems can be solved by a team of such robots.

Crucial for the solvability of a problem is the activation schedule of robots and the duration of their activities in each cycle. Three main settings have been considered. In the *fully synchronous* setting, $\mathcal{FSYNC}$, time is discrete, all robots are active at all times, and cycles are to be considered instantaneous. The *semi-synchronous* setting, $\mathcal{SSYNC}$, is like the fully synchronous one except that at each time only a subset of the robots are active; the choice of which robots are active at which time is made by an adversary. In the *asynchronous* setting, $\mathcal{ASYNC}$, there is no common notion of time (which is possibly continuous), the times when each robot is activated as well as the duration of each activity is decided by an adversary for each cycle.

Both in $\mathcal{SSYNC}$ and $\mathcal{ASYNC}$, the adversarial scheduler is constrained to be fair; i.e., for each robot r and time t there is a time $t' > t$ at which r will be active.

Another important factor is whether the movements are *rigid*; that is when performing the *Move* operation, a robot always reaches its destination, regardless of the distance. When movements are *non-rigid*, the *Move* operation can be interrupted before its completion by an adversary whose only limit is that, if the robot does not reach its destination, it moves least a distance $\rho > 0$, unknown to the robots.

The study of which problems can be solved by a team of such robots has been conducted in all three settings, considering all those factors. The main focus has been on the important class of *Pattern Formation* problems, where the robots are required to arrange themselves to form a given geometric shape (e.g., [4, 5, 10, 12, 18–20]). Interestingly, this class includes the *Point Formation* problem requiring the robots to move to the same location, not decided in advance. This problem, known also as *Gathering* or *Rendezvous*, is of particular importance and has been extensively studied in all three settings (e.g., [2, 11, 13, 14]). Other investigated problems include *Election* (e.g., [7]) *Flocking* (e.g., [15, 17, 21]), *Scattering* and *Spreading* (e.g., [1, 3, 6, 16]), etc. For more details, see the monograph [9] and the recent book (and chapters therein) [8].

The crucial point is that all these investigations consider a *single* team performing the same task. Nobody has yet considered the case of two (or more) teams of robots, in the same space at the same time, each team with a different goal, needing to solve a different problem.

Clearly each robot knows the problem it must solve, but because they all are outwardly undistinguishable, it does not know which are the other members of its team. In this novel setting, several questions immediately arise. In particular: Is it possible for each of the two teams to solve its problem ? Can it be done by the same protocol ? Under which setting ? Under what conditions ? What happens if the two tasks to be performed are conflicting ?

1.2 Contributions

In this paper we start this line of investigation by considering two conflicting classical pattern formation problems, and we show how they can be solved concurrently by two anonymous teams operating in the same space under the worst adversarial scheduler (asynchronous and with non-rigid movements).

More precisely, we are given two teams of anonymous robots $\mathcal{R}_g$ and $\mathcal{R}_f$, where the goal of the robots in $\mathcal{R}_g$ is to achieve *Gathering* (i.e., to meet at some arbitrary point), while the goal of the robots in $\mathcal{R}_f$ is to solve the *Circle Formation* problem (i.e., to place themselves at distinct points of an arbitrary circle). The robots are located at distinct points in the same space and they do not know who, among the robots they see, belong to their own team.

We devise a terminating deterministic algorithm that allows each team to solve its problem asynchronously and with non-rigid movements.

Our algorithm assumes agreement of the local coordinate axes (but not necessarily of their orientation) and weak multiplicity detection (i.e., the ability to distinguish points occupied by more than one robot).

2 MODEL AND PRELIMINARIES

2.1 Model

Let $\mathcal{R}_g$ and $\mathcal{R}_f$ be two teams of robots in the Euclidean plane operating in Look-Compute-Move cycles in the standard $\mathcal{OBLOT}$ model [8]. Let $\mathcal{R} = \{r_1, r_2, \dots, r_n\} = \mathcal{R}_g \cup \mathcal{R}_f$, with $|\mathcal{R}_g| \geq 6$ and $|\mathcal{R}_f| \geq 2$.

The robots are represented by points and they can move freely in the plane; they are anonymous, indistinguishable by their appearances, and they run the same distributed algorithm; they do not have explicit means of communication; they are oblivious in the sense that, at the beginning of each Look-Compute-Move cycle, a robot does not carry any information from its previous cycles (i.e., whenever robots are activated they start afresh). The robots lack a global coordinate system, they however share common axis (without necessarily agreeing on their directions); each robot has its own axis direction having origin at its current position. Robots have *weak multiplicity detection* capability which enables them to determine whether there is more than one robot at single point.

Within the $\mathcal{OBLOT}$ model, we consider two strong adversarial conditions. First of all, the setting is $\mathcal{ASYNC}$: there is no common notion of time and, except for the *Look* operation that is instantaneous, all other operations take a finite but arbitrary amount of time, decided by the adversary. Furthermore, the movements are *non-rigid*: every movement performed in the *Move* operation can be interrupted by an adversary; if the robot does not reach its destination, however, it travels at least a distance $\rho > 0$, unknown to the robots.

The robots in $\mathcal{R}_g$ and $\mathcal{R}_f$ are required to solve the *gathering problem* (i.e., to gather exactly at some point) and the *circle formation problem* (i.e., to place themselves on a circle), respectively. A robot knows the team to which it belongs, however it can not distinguish the teams the other robots belong to. We assume that initially, at time $t = 0$, all robots occupy distinct positions and they are stationary.

2.2 Preliminaries

Let $r_i(t)$ denote the point occupied by robot r_i at time t. A robot configuration is denoted by the multiset $\mathcal{R}(t) = \{r_1(t), \dots, r_n(t)\}$. Let $\mathcal{R}_g(t)$ and $\mathcal{R}_f(t)$ denote the set of robot positions in $\mathcal{R}_g$ and $\mathcal{R}_f$, respectively, at time t.

A *multiplicity point* is a point where at least two robots lie. A *stable multiplicity point* contains at least two robots from $\mathcal{R}_g$.

Let $S(t)$ denote the smallest enclosing circle of the points in $\mathcal{R}(t)$ and let O be the center of the initial smallest enclosing circle $S(0)$. Moreover, let $S_{out}(t)$ and $S_{in}(t)$ denote the robot positions in $\mathcal{R}(t)$ lying on $S(0)$ and inside $S(0)$, respectively, at time t.

We define a sequence of concentric circles $C_1(t), C_2(t) \dots C_k(t)$ centered in O, as follows. Let $C_i(t)$ be the circle centered at O and passing through the i^{th}-nearest robot positions from O, (not lying at O) for $1 \leq i \leq k$ (see Figure 1). Note that, if $k = 1$ (i.e., there is just one circle $C_1(t) = C_k(t) = S(0)$), then either $|S_{in}(t)| = 0$ or all the robots, having positions in $S_{in}(t)$, lie at O.

Let p and q be two points in the plane. By (p, q) and $\overline{pq}$, we denote the open (excluding p and q) and closed (including p and q) line segments joining p and q, respectively.

Pivotal Robot Positions.

We define a special set of robots positions $\mathcal{P}(t) \subset S_{out}(t)$ (the *pivotal robot positions*) which are sufficient to define $S(0)$. The robot positions in $\mathcal{P}(t)$ are used to maintain $S(0)$ during part of the algorithm execution.

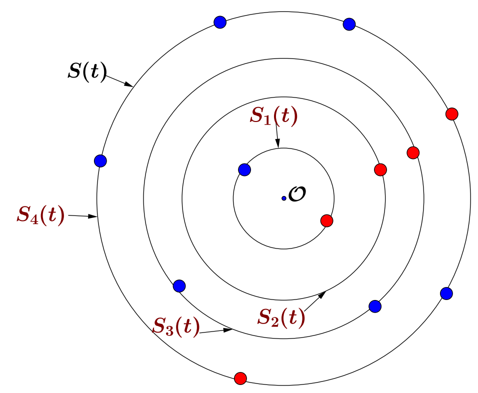

Figure 1: Illustrations of the circles $C_k(t)$, $k \geq 1$: blue robots are from $\mathcal{R}_f$ and red robots are from $\mathcal{R}_g$

If $|S_{out}(t)| \leq 4$, then $\mathcal{P}(t) = S_{out}(t)$.

If $|S_{out}(t)| > 4$, let $\mathcal{L}$ be the line passing through O and parallel to the Y-axis, intersecting $S(t)$ in two points p_1 and p_2. The set $\mathcal{P}(t)$ is defined differently depending on the situation:

(1) Exactly one of p_1 and p_2 contains a robot position (without loss of generality, let it be p_1 containing robot position $r_c(t)$ (see Figure 2(A)). Let $r_d(t)$ and $r_e(t)$ be the two robot positions in $S_{out}(t)$ such that they lie on different sides of $\mathcal{L}$ and closest to p_2. Then $\mathcal{P}(t) = \{r_c(t), r_d(t), r_e(t)\}$.

(2) Both p_1 and p_2 contain robot positions (Figure 2(B)). Let $r_i(t)$ and $r_j(t)$ be two robot positions at p_1 and p_2 respectively. Then $\mathcal{P}(t) = \{r_i(t), r_j(t)\}$.

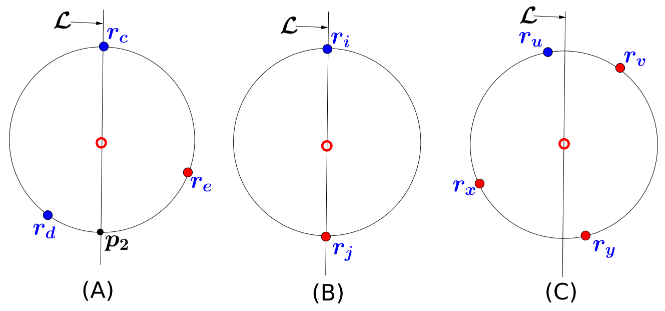

Figure 2: Illustrations of different scenarios of pivotal section: (A) exactly one of p_1 and p_2 contains a robot position (B) both of p_1 and p_2 contain robot positions (C) none of p_1 and p_2 contains a robot position. Blue robots are from $\mathcal{R}_f$ and red robots are from $\mathcal{R}_g$

(3) Neither p_1 nor p_2 contain a robot position (Figure 1(C)). Let $r_u(t)$ and $r_v(t)$ be two robot positions in $S_{out}(t)$ such that they lie on two different sides of $\mathcal{L}$ and they are clockwise and counterclockwise neighbors of p_1 on $S(t)$. Similarly, let $r_x(t)$ and $r_y(t)$ be two robot positions in $S_{out}(t)$ such that they lie on two different sides of $\mathcal{L}$ and they are clockwise and counterclockwise neighbors of p_2 on $S(t)$. In this case, $\mathcal{P}(t) = \{r_u(t), r_v(t), r_x(t), r_y(t)\}$.

Lemma 1. *The set of points $\mathcal{P}(t)$ uniquely defines the smallest enclosing circle $S(t)$.*

3 THE ALGORITHM

3.1 Overview and Preliminaries

During the algorithm, the initial smallest enclosing circle $S(0)$ plays an important role and the algorithm is designed in such a way that $S(0)$ is always recognizable.

First of all, robots in $\mathcal{R}_g$ coordinate their movements to create a multiplicity point at the center of $S(0)$ (*Multiplicity Creation Phase*). In this phase, if necessary, we make use of the agreement on the axis directions to select special robots (the ones on pivotal positions) for preserving $S(0)$; in fact, during this phase $S(t)$ always coincides with S(0).

Once a multiplicity point is created, the robots start the *Formation Phase*, during which the robots in $\mathcal{R}_g$ move towards the multiplicity point and those in $\mathcal{R}_f$ move towards $S(0)$. All movements are designed in such a way that the original multiplicity point is preserved throughout the execution of the algorithm in spite of asynchrony and non-rigidity. Note that, in this phase, the smallest enclosing circle might change. However, the robots can always reconstruct $S(0)$ by computing the circle centered at the multiplicity point and passing through the farthest robot position from it.

This simple structure requires a careful design of the rules as the indistinguishability and the asynchrony of the robots pose several difficulties. Two such difficulties are: 1) guaranteeing that the multiplicity point that is initially formed remains unchanged, and 2) ensuring that no new multiplicity points are created. The first issue is related to the possibility of disintegration of the multiplicity point formed by a robot in $\mathcal{R}_f$, being initially in the center of $S(0)$, and one in $\mathcal{R}_g$ arriving there; the second issue is related to collisions. Our algorithm guarantees that both problems are avoided.

Note that, since robots are asynchronous and indistinguishable, the multiplicity creation phase and the formation phase may run concurrently (i.e., some robots might start executing the second phase while others are still executing the first one) creating additional potential problems. The rules of the two phases must thus be compatible in order to maintain the continuity of the multiplicity point and to avoid collisions in spite of the concurrent executions.

In the algorithm description, we will consider the following classes of configurations:

- $\mathcal{M}ulti$: Configurations having exactly one multiplicity point.
- $\mathcal{F}ewOut$: Configurations having no multiplicity points, with $|S_{out}(t)| \leq 4$.
- $\mathcal{M}anyOut$: Configurations having no multiplicity points, with $|S_{out}(t)| > 4$.

Observe that, since $|\mathcal{R}_g| \geq 6$, for a configuration $\mathcal{R}(t) \in \mathcal{F}ewOut$, the set $S_{in}(t)$ contains at least two members from $\mathcal{R}_g(t)$. Also note that, by design, we will ensure that these classes form a partition of all possible configurations arising during the execution of the algorithm.

3.2 Algorithm PatternFormation()

In both phases, movements of the robots are regulated by a special routine, *MoveToDestination()*, which will be described later, whose objective is to maintain some form of synchronization in the robots' movements in order to guarantee the absence of collisions (and thus prevent potential creation of multiple multiplicity point).

3.2.1 Multiplicity creation phase. This phase is executed when there are no multiplicity points in the current configuration and the goal is to create a unique multiplicity point at O. In order to create a multiplicity point, robots lying on $S(t)$ that are not in pivotal positions are let to move inside of $S(t)$ (regardless of the team they belong to) until $|S_{out}(t)| \leq 4$. During this time, on the other hand, robots lying inside $S(t)$ do not move. Since $|\mathcal{R}_g| \geq 6$ this procedure ensures that, before robots start moving to create the multiplicity point, there are at least two robots from the set $\mathcal{R}_g$ which lie inside $S(t)$. These robots will eventually make the multiplicity point stable.

More precisely, robot r_i acts as follows, according to its position and the class to which $\mathcal{R}(t)$ belongs.

(1) $\mathcal{R}(t) \in \mathcal{F}ewOut$:
- $r_i(t) \in S_{out}(t)$: Robot r_i does not move regardless of the team it belongs to. This ensures that $S(t)$ remains invariant until a stable multiplicity point is created at O.
- $r_i(t) \in S_{in}(t) \cap \mathcal{R}_f(t)$: Robot r_i moves, according to algorithm *MoveToDestination()*, towards the circle $C_{k-1}(t)$ (where $C_k(t) = S(t)$). The movement of these robots ensures free corridors for robots in $\mathcal{R}_g$ towards O to avoid the creation of multiple multiplicity points while maintaining $|S_{out}(t)| \leq 4$.
- $r_i(t) \in S_{in}(t) \cap \mathcal{R}_g(t)$: Robot r_i moves towards O according to algorithm *MoveToDestination()*. Note that, since $|S_{out}(t)| \leq 4$ and $|\mathcal{R}_g| \geq 6$, the set $S_{in}(t) \cap \mathcal{R}_g(t)$ contains at least two robot positions which will eventually create a stable multiplicity point.

(2) $\mathcal{R}(t) \in \mathcal{M}anyOut$: In this case, $S_{out}(t) > 4$. Robots lying on $S(t)$ (except for those on the pivotal points $\mathcal{P}(t)$), move inside $S(t)$ as follows to convert the current configuration into one in $\mathcal{F}ewOut$.
- $r_i(t) \in S_{out}(t)$: If $r_i(t) \notin \mathcal{P}(t)$, r_i moves towards its destination point $p_i(t)$ computed as follows: if $C_1(t) = S(t)$, then $p_i(t)$ is the middle point of $\overline{r_i(t)O}$; otherwise, then $p_i(t)$ is the middle point of $\overline{r_i(t)w_i}$, where w_i is the intersection point between the circle $C_{k-1}(t)$ and $rad_i(t)$. On the other hand, if $r_i(t) \in \mathcal{P}(t)$, robot r_i does not move.
- $r_i(t) \in S_{in}(t)$: Robot r_i does not move.

3.2.2 Formation phase. This phase is executed only after a multiplicity point has been created at O; that is, when $\mathcal{R}(t) \in \mathcal{M}ulti$. We denote the multiplicity point by p_m. During this phase, the robots form their assigned pattern. We ensure that at least one robot from $\mathcal{R}_f$ lies on $S(0)$ before the robots in $\mathcal{P}(t) \cap \mathcal{R}_g$ start moving; this ensures that all robots can compute $S(0)$ using the multiplicity point p_m and the farthest robot position from the multiplicity point.

More precisely, robot r_i acts according to its position as follows.

- $r_i(t) \in S_{out}(t) \wedge |S_{in}| > 1$: Robot r_i does not move. This is required to allow the robots to be able to reconstruct $S(0)$.
- $r_i(t) \in S_{out}(t) \wedge |S_{in}| = 1$: If $r_i \in \mathcal{R}_f$, it does not move. Otherwise, it moves towards p_m along the line segment $\overline{r_i(t)p_m}$.
- $r_i(t) \in S_{in}(t) \cap \mathcal{R}_f(t)$: Robots compute $S(0)$ as the circle centered at p_m and passing through the farthest robots position in $\mathcal{R}(t)$ from p_m. If $r_i(t) = O$ and $|S_{in}(t)| > 1$, r_i does not move (this is required for the stability of the multiplicity point); otherwise, r_i moves towards $S(0)$ according to algorithm *MoveToDestination()*.
- $r_i(t) \in S_{in}(t) \cap \mathcal{R}_g(t)$: Robot r_i moves towards O according to algorithm *MoveToDestination()*.

Note that the two phases might very well run concurrently. In fact, once a multiplicity point is created, some robots may still be executing the multiplicity creation phase while some might start the formation one.

3.3 Routine MoveToDestination()

Routine *MoveToDestination()* prescribes the robots' movement in such a way that during the whole execution of *PatternFormation()* exactly one multiplicity point is created and robots reach their respective destination points in finite time.

The algorithm achieves some form of movement sequentialization by allowing only robots on the smallest concentric circle C_1 to move: the movement is either straight towards the center (in the case of robots in $\mathcal{R}_g$) or towards C_2 at some appropriate angular direction to create/maintain a free corridor to $S(0)$ (in the case of robots in $\mathcal{R}_f$). In doing so the number of concentric circles decreases while free corridors are created, and eventually all robots will be either on $S(0)$ or at O.

More precisely, robot r_i acts as follows depending on its position.

- **r_i lies on $C_1(t)$:**
If $r_i \in \mathcal{R}_g$, then r_i has O as its destination point, and it moves straight to O along $\overline{r_i(t)O}$.
If $r_i \in \mathcal{R}_f$, then r_i moves towards $C_2(t)$ chosing a special angular direction. Let $rad_i(t)$ intersect the circle $C_2(t)$ at the point q_i. If $rad_i(t)$ does not contain any other robot position, then robot r_i moves to q_i along $rad_i(t)$. Otherwise, it moves in the following way: Let $A_i(t)$ be the set of all robot positions in $\mathcal{R}(t)$ not lying on $\overline{r_i(t)p_i}$. Let $r_j(t) \in A_i(t)$ be such that $\theta_i(t) = \angle r_i(t)Or_j(t)$ is maximum for all robot positions in $A_i(t)$ (tie, if any, broken arbitrarily). Let x be a point on the circle $C_1(t)$ such that:
$$\angle r_i(t)Ox = \frac{1}{3}\theta_i(t)$$
Let $\mathcal{L}_i$ be the half-line originated at O and passing through x. Let $\mathcal{L}_i$ intersect $C_2(t)$ at the point u_i.
Robot r_i moves towards u_i along $\overline{r_i(t)u_i}$ (see Figure 3).
- **r_i lies at O and $r_i \in \mathcal{R}_f$:**
In this case, robot r_i moves towards $S(0)$ only when all the robots inside $S(t)$ lie at O. Let $\mathcal{L}^*(t)$ be the bisector of the angle the largest angle made by two consecutive robot positions on $S(t)$. Let v_i be the intersection point between $\mathcal{L}^*(t)$ and $S(t)$. Robot r_i moves towards v_i along $\overline{r_i(t)v_i}$.

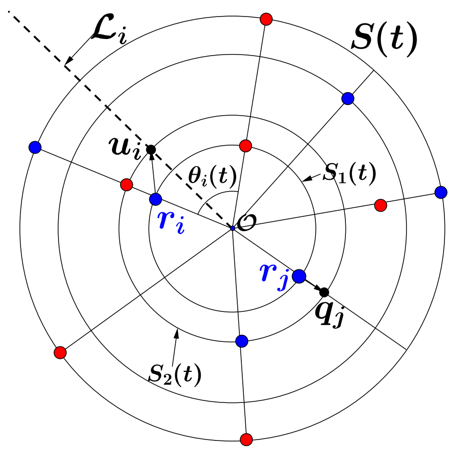

Figure 3: Illustration of movements of the robots during algorithm *MoveToDestination()*: **robot r_i moves towards the point u_i lying on $C_2(t)$ and robot r_j move towards the point q_j lying on $C_2(t)$**

- **All other cases**: r_i does not move.

4 CORRECTNESS

We now show that our solution is correct by proving that a unique point p_m is created and that, within finite time, robots in $\mathcal{R}_g$ gather in p_m, while robots in $\mathcal{R}_f$ position themselves on the original smallest enclosing circle $S(0)$.

Lemma 2. *Routine MoveToDestination() provides collision free movements for the robots.*

Proof. We prove the lemma by considering each case separately. *For robots executing the Multiplicity Creation Phase:* Consider first the movements of the robots when $\mathcal{R}(t) \in \mathcal{M}any\mathcal{O}ut$. In this case, a robot r_i moves only if it is on $S(t)$ and do so along $rad_i(t)$ to a destination point which lies on $rad_i(t)$ and is different O. Since the destination points of any two such moving robots are distinct, so are their radii, hence their movements are collisions free.

Consider now the case when $\mathcal{R}(t) \in \mathcal{F}ew\mathcal{O}ut$. By construction, the movements of the robots that perceive $\mathcal{R}(t) \in \mathcal{F}ew\mathcal{O}ut$ are ordered according to their distances from O, and only the robots on $C_1(t)$ move.

The robots in $\mathcal{R}_g$ move straight towards O and their paths do not intersect; thus their movements are collisions free.

Consider the robots in $\mathcal{R}_f$. By routine *MoveToDestination()*, these robots move towards $C_2(t)$. Consider two robots $r_i, r_j \in \mathcal{R}_f$ lying on $C_1(t)$. We show that the paths of these robots towards their respective destination points do not intersect in between. If these robots have robots-free straight paths towards $C_2(t)$ along $rad_i(t)$ and $rad_j(t)$ respectively, then they move along $rad_i(t)$ and $rad_j(t)$ and they clearly do not collide with each other. Now, suppose that neither of them has a free corridor to $C_2(t)$ (the case when exactly one of them has a free corridor, follows from this case). Let the wedge $\mathcal{D}_{ij}(t)$, defined by $rad_i(t)$ and $rad_j(t)$, contain at least one robot, say r_k, inside it (Figure 4(A)). Then by routine *MoveToDestination()*, the paths of r_i and r_k are separated by $rad_k(t)$ and they do not collide. Suppose $\mathcal{D}_{ij}(t)$ does not contain any robot position inside it (Figure 4(B)). If at least one robot has a path lying outside of $D_{ij}(t)$, then these two robots do not collide. Otherwise, $\angle r_i(t)Or_j(t) = \theta_i(t)$. Let $\mathcal{L}_i^b(t)$ be the bisector of the angle $\theta_i(t)$. According to routine *MoveToDestination()*, the paths of the robots r_i and r_j are separated by $\mathcal{L}_i^b(t)$.

Note that, since robots are asynchronous, some robots may have *pending movements* from the case $\mathcal{R}(t) \in \mathcal{M}any\mathcal{O}ut$. These movements, however are towards the interior of $S(t)$ (stopping before $C_{k-1}(t)$) and do not interfere with the other movements.

We can conclude that the lemma holds in this case.

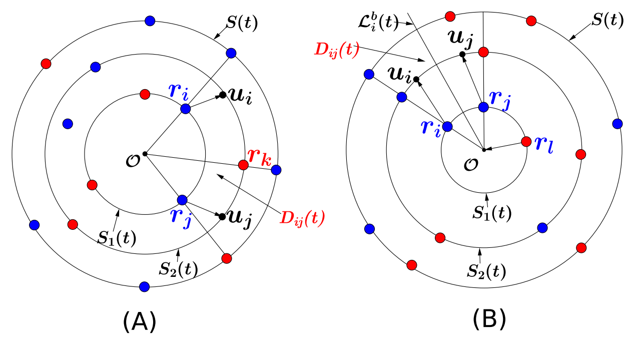

Figure 4: Illustration of scenarios for the proof of lemma 2: (A) wedge $\mathcal{D}_{ij}(t)$ contains a robot position $r_k(t)$ (B) $\mathcal{D}_{ij}(t)$ does not contain any robot position inside it

For robots executing the Formation Phase: If $r_l \in \mathcal{R}_f$ does not lie at O, according to routine *MoveToDestination()*, its movements are the same as if in the multiplicity creation phase. If instead $r_l \in \mathcal{R}_f$ lies at O (which can happen only if it was there intitially at time $t = 0$), according to routine *MoveToDestination()*, it moves only when all robots in $\mathcal{R}_f$ are on $S(0)$. When this happens, only some robots in $\mathcal{R}_g$ lying on $S(0)$ might concurrently move towards O (these must be the robots which were initially on pivotal positions). These robots move straight towards O and robot r_l moves along a free corridor to $S(0)$. These paths do not intersect by construction; thus, they are collision free.

□

Lemma 3. *The multiplicity creation phase creates a unique multiplicity point, which becomes stable in finite time.*

Proof. Consider a configuration $\mathcal{R}(t) \in \mathcal{F}ew\mathcal{O}ut \cup \mathcal{M}any\mathcal{O}ut$. If $\mathcal{R}(t) \in \mathcal{M}any\mathcal{O}ut$, robots coordinate their movement to convert $\mathcal{R}(t)$ into a configuration in $\mathcal{F}ew\mathcal{O}ut$. In this case, robots lying on $S(t)$ are moved inside $S(t)$ in such way that $S(t)$ remains invariant (which is possible by Lemma 1 and the fact that $|\mathcal{R}_g| \geq 6$). Thus $\mathcal{R}(t)$ is converted into a configuration in $\mathcal{F}ew\mathcal{O}ut$ in finite time.

Suppose $\mathcal{R}(t) \in \mathcal{F}ew\mathcal{O}ut$. We have to show that at least two robots from the set $\mathcal{R}_g$ reach O in finite time. The movements of the robots are ordered according to their distances from O. A robot is eligible for movement at time t only if it lies on $C_1(t)$. If $C_1(t)$ contains at least two robots from $\mathcal{R}_g$, then these robots reach O in finite time and we have a stable multiplicity point. Otherwise, the robots in $\mathcal{R}_f$, lying on $C_1(t)$, move towards $C_2(t)$. These movements convert $C_2(t)$ to $C_1(t')$ in finite time, for some $t' > t$ and the robots lying on $C_1(t')$ become eligible for movements. If initially the circles $C_1(t)$ and $C_2(t)$ contain at least two robots from $\mathcal{R}_g$, then we are done. Otherwise, the robots from $C_1(t')$ move towards $C_2(t')$. This process continues until at least two robots from $\mathcal{R}_g$ find themselves eligible for movements. Since the number of robots is finite and $|\mathcal{R}_g| \geq 6$, this process terminates (in the worst case, robots in $\mathcal{R}_f$ may have to move to $C_{k-1}(t)$ where $C_k(t) = S(t)$). This implies that within finite time at least two robots from $\mathcal{R}_g$ reach O. The uniqueness of the multiplicity point follows from Lemma 2. □

Lemma 4. *Algorithm PatternFormation() solves the gathering problem for the robots in $\mathcal{R}_g$ within finite time.*

Proof. We have to show that all robots in $\mathcal{R}_g$ gather at a single point. By Lemma 3, a unique multiplicity point is created at O during the multiplicity creation phase and this point becomes stable in finite time. Since robots are endowed with weak multiplicity detection capability, they can identify the multiplicity point. The robots in $\mathcal{R}_g$ move towards the multiplicity point according to routine *MoveToDestination()*. By Lemma 2 and the same argument as in the proof of Lemma 3, we can conclude that within finite time all robots in $\mathcal{R}_g$ reach O. □

Lemma 5. *Algorithm PatternFormation() solves the circle formation problem for the robots in $\mathcal{R}_f$ within finite time.*

Proof. The robots in $\mathcal{R}_f$ solves the circle formation problem by placing themselves on $S(0)$. Since robots have weak multiplicity detection capability and exactly one multiplicity point is created, robots can identify O once it becomes a multiplicity point. If $\mathcal{P}(t)$ contains at least one robot position from $\mathcal{R}_f(t)$, then the multiplicity point and this robot uniquely defines $S(0)$: the circle having center at the multiplicity point and passing through the farthest robot position from the multiplicity point. Suppose $\mathcal{P}(t)$ contains no robot position from $\mathcal{R}_f(t)$. In this case, the robots having position in $\mathcal{P}(t)$ move, by construction, only when $|S_{in}(t)| = 1$. During the whole execution of algorithm *PatternFormation()*, robots in $\mathcal{R}_f$ do not have O as their destination point. Since $|\mathcal{R}_f| \geq 2$, we can say that $S(t)$ contains at least one robot from $\mathcal{R}_f$ when the robots having positions in $\mathcal{P}(t)$ start moving. Since robots move according to routine *MoveToDestination()*, we can conclude the proof by Lemma 2. □

From the above sequence of lemmas, we have:

Theorem 1. *Algorithm PatternFormation() allow the set of robots in $\mathcal{R}_f$ to form a circle and the set of robots $\mathcal{R}_g$ to gather within finite time.*

5 CONCLUSIONS

In this paper, we have started a new line of investigation within the context of robots operating according to the Look-Compute-Move model. In fact, we have considered for the first time the simultaneous presence in the same space of more than one team of undistinguishable robots, each team with a different task to solve. In particular, we have considered two teams and two conflicting tasks: gathering and circle formation. The solution we proposed works under the worst possible adversarial scheduler (asynchronous and non-rigid), but assumes agreement of the local coordinate axes (not necessarily of their orientation) and weak multiplicity detection.

This result opens several interesting investigation directions. Apart from the obvious open problem of relaxing the assumptions, the study of various combinations of tasks constitutes a whole new area of investigation worth exploring.

ACKNOWLEDGMENTS

This research has been supported in part by the Natural Sciences and Engineering Research Council of Canada (N.S.E.R.C.) under the Discovery Grant program, and by Dr. Flocchini's University Research Chair. This work has been done while S. Bhagat was a postdoctoral fellow under the supervision of P. Flocchini and N. Santoro.

REFERENCES

[1] D. Canepa, X. Défago, T. Izumi, and M. Potop-Butucaru. Flocking with oblivious robots. In *18th Int. Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 94–108, 2016.
[2] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 2012.
[3] R. Cohen and D. Peleg. Local spreading algorithms for autonomous robot systems. *Theoretical Computer Science*, 399:71–82, 2008.
[4] S. Das, P. Flocchini, N. Santoro, and M. Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. *Distributed Computing*, 28(2):131–145, 2015.
[5] Y. Dieudonné, O. Labbani-Igbida, and F. Petit. Circle formation of weak mobile robots. *ACM Transactions on Autonomous and Adaptive Systems*, 3(4):16:1–16:20, 2008.
[6] Y. Dieudonné and F. Petit. Scatter of robots. *Parallel Processing Letters*, 10(1):175–184, 2009.
[7] Y. Dieudonné, F. Petit, and V. Villain. Leader election problem versus pattern formation problem. In *International Symposium on Distributed Computing (DISC)*, LNCS 6343, pages 267–281, 2010.
[8] P. Flocchini, G. Prencipe, and N. Santoro (Eds). *Distributed Computing by Mobile Entities*. Springer, 2019.
[9] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Morgan & Claypool Publishers, 2012.
[10] P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta. Distributed computing by mobile robots: uniform circle formation. *Distributed Computing*, 30(6):413–457, 2017.
[11] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous mobile robots with limited visibility. *Theoretical Computer Science*, 337:147–168, 2005.
[12] 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.
[13] P. Flocchini, N. Santoro, G. Viglietta, and M. Yamashita. Rendezvous with constant memory. *Theoretical Computer Science*, 621:57–72, 2016.
[14] S. Gan Chaudhuri and K Mukhopadhyaya. Leader election and gathering for asynchronous fat robots without common chirality. *Journal of Discrete Algorithms*, 33:171–192, 2015.
[15] V. Gervasi and G. Prencipe. Coordination without Communication: The Case of the Flocking Problem. *Discrete Applied Mathemathics*, 144(3):324–344, 2004.
[16] T. Izumi, D. Kaino, M. Potop-Butucaru, and S. Tixeuil. On time complexity for connectivity-preserving scattering of mobile robots. *Theor. Comput. Sci.*, 738:42–52, 2018.
[17] S. Souissi, Y. Yang, X. Défago, and M. Takizawa. Fault-tolerant flocking for a group of autonomous mobile robots. *The Journal of Systems and Software*, 84:29–36, 2011.
[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. Yang, N. Xiong, N. Y. Chong, and X. Défago. A decentralized and adaptive flocking algorithm for autonomous mobile robots. In *The 3^{rd} International Conference on Grid and Pervasive Computing Workshops*, pages 262–268, 2008.