Gathering Autonomous Mobile Robots in Non Totally Symmetric Configurations

TECHNICAL REPORT no. 379
ETH Zurich, Department of Computer Science

Mark Cieliebak
ETH Zurich, Switzerland
cieliebak@inf.ethz.ch

Giuseppe Prencipe
Università di Pisa, Italy
prencipe@di.unipi.it

Abstract

We study the problem of coordinating a set of autonomous mobile robots that can freely move in a two-dimensional plane; in particular, we want them to gather at a point not fixed in advance (GATHERING PROBLEM). We introduce a model of weak robots (decentralized, asynchronous, no common knowledge, no identities, no central coordination, no direct communication, oblivious) which can observe the set of all points in the plane that are occupied by other robots. Based on this observation, a robot uses a deterministic algorithm to compute a destination, and moves there. The problem is unsolvable if the robots have no additional abilities. Therefore, we introduce the ability to detect how many robots are at a specific point in the plane (*multiplicity detection*). For two robots, the problem remains unsolvable. For 3 and 4 robots, we give algorithms that solve the GATHERING PROBLEM. For more than 4 robots, we present an algorithm that gathers the robots if they are not in a specific symmetric configuration at the beginning (*totally symmetric configuration*). We show how to solve such initial configurations separately. However, the general solution of the GATHERING PROBLEM remains an open problem.

Keywords: Autonomous Mobile Robots, Gathering, Distributed Coordination, Symmetric Configurations.

1 Introduction

We consider a distributed system whose entities are autonomous mobile robots modeled as devices with computational capabilities that are able to freely move in a two-dimensional plane. We study the problem of coordinating these robots. The coordination mechanism is totally *decentralized*, i.e., the robots are completely *autonomous* and no central control is used. The objective is to gather all robots at one point. This point is not fixed in advance.

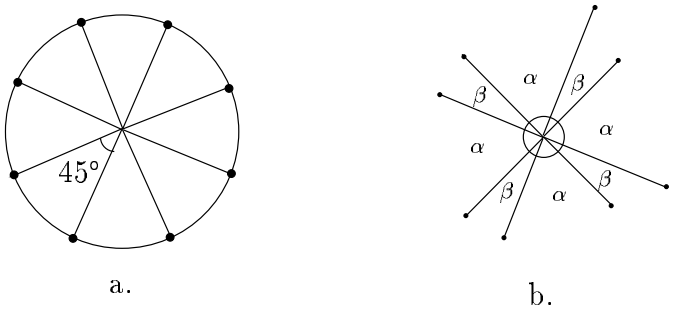
The GATHERING PROBLEM is one of the basic interaction primitives studied in a system populated by a set of autonomous mobile robots [8]. The problem of coordinating a collection of autonomous mobile robots has been studied in robotics and in artificial intelligence [2, 7, 8]. Mostly, the problem is approached from an experimental point of view: algorithms are designed using mainly heuristics, and then tested either by means of computer simulations or with real robots. Neither proofs of correctness of the algorithms, nor any analysis of the relationship between the problem to be solved, the capabilities of the robots employed, and the robots' knowledge of the environment are given. Recently, concerns on computability and complexity of the coordination problem have motivated *algorithmic* investigations, and the problem has also been approached from a *computational* point of view [1, 5, 6, 10, 11, 13]. In [1] and [13], any action of the robots, including moving, is *instantaneous*, while in [5, 6, 10, 11], as well as in this paper, there is no such assumption. An extended abstract of this paper has been published in [3].

We consider a very weak model of robots: the robots are anonymous, have no common knowledge, no central coordination, and no means of direct communication. Initially, they are in a waiting state. They wake up asynchronously, observe the other robots' positions, compute a point in the plane, move towards this points (but may not reach it[1]), and become waiting again.

These robots cannot gather if they have no additional abilities: a necessary ability that allows them to solve the GATHERING PROBLEM is to detect multiplicities, i.e., to be able to detect whether there is more than one robot at the same point. This ability is used as follows: first let at least two robots meet at a point, so that there is a point in the plane with multiplicity greater than one; then, all remaining robots move to this point (thereby, they avoid to generate another point with multiplicity greater than one).

For less than five robots, there exist simple algorithms to gather the robots at a point. The problem becomes challenging for $n \geq 5$ robots: how can we gather a large set of robots? One idea would be to gather the robots at the Weber point. The *Weber point* of a set of distinct points is the unique point in the plane that minimizes the sum of the distances between itself and all points in the given set of points [14]. An interesting property of the Weber point is that it does not change when moving any of the points straight towards it: hence, the robots could simply gather at their Weber point. Unfortunately, there exist point sets such that the

[1]That is, a robot can stop before reaching its destination point, e.g. because of limits to the robot's motion energy.



a. b.

Figure 1: (a) A totally symmetric configuration. (b) A biangular configuration.

Weber point is not computable [4]. Therefore, we need a different strategy. Assume that the initial positions of the robots form a regular n-gon, i.e., all robots are on a circle and the distances between each two adjacent robots are equal: we call this a *totally symmetric* configuration (see Figure 1.a). Regular n-gons are a special case of biangular configurations: the robots are in *biangular configuration with center* c if there exists an ordering of the robots and two angles α and β such that all angles between two adjacent robots w.r.t. c are either α or β, and the angles alternate (see Figure 1.b). Biangular configurations (hence, totally symmetric) can be identified in finite time. If the robots are at the beginning in biangular configuration (this clearly include also the configurations that are totally symmetric), we can use the following simple strategy: all robots move straight towards c. The robots remain in biangular configuration (or in a degenerated variant where one robot is at c) until at least two of them have reached c. Then, we have multiplicity greater than 1 at c, and all robots will gather there.

On the other hand, if the initial configuration is not totally symmetric, we can design a different (and rather sophisticated) algorithm for the GATHERING PROBLEM: first, we compute the smallest circle enclosing all robots; this circle is unique. Then we elect a strict subset of the robots such that the smallest enclosing circle does not change when we move these robots towards the center of the circle. As soon as two robots have reached the center of the circle, we have a point with multiplicity greater than one, and again all robots will gather there.

The remaining challenge is to combine these two cases, i.e., to design an algorithm that solves the GATHERING PROBLEM for *any* initial configuration.

The rest of the paper is organized as follows. In the next section, we define the model of robots we are using, the problem to solve, and some notation. In Section 3, we provide solutions for the GATHERING PROBLEM for $n = 2, 3$, and 4 robots. In Sections 4 and 5 we present the two algorithms for the GATHERING PROBLEM for arbitrary $n \geq 5$, one for biangular initial configurations and one for initial configurations that are not totally symmetric. Conclusions are drawn in Section 6.

2 Model and Definitions

2.1 Autonomous Mobile Robots

Each robot is viewed as a point, and it is equipped with sensors that let it observe the set of all points in the plane that are occupied by at least one other robot, and form its *local view* of the world. Note that a robot only knows *whether* there are other robots at a specific point, but it has no knowledge about their *number*. The local view of each robot includes a unit of length, an origin (which we assume w.l.o.g. to be the position of the robot in its current observation), and a coordinate system (e.g. Cartesian). We do not assume any kind of agreement among the robots on the unit of length, the origin, or the local coordinate systems.

A robot is initially in a *waiting* state (*Wait*). Asynchronously and independently from the other robots, it *observes* the environment (*Look*) by activating its sensors. The sensors return a snapshot of the world, i.e., the set of all points that are occupied by at least one other robot, with respect to the local coordinate system. The robot then *calculates* its destination point (*Compute*) according to its deterministic algorithm, based only on its local view of the world. Each robot executes the same deterministic algorithm. It then *moves* towards the destination point (*Move*); if the destination point is the current location, the robot stays still. After an unpredictable time (during or after the move), the robot returns to the waiting state. Therefore, it may or may not reach its destination point during the move. The sequence *Wait* - *Look* - *Compute* - *Move* forms a *cycle* of a robot.

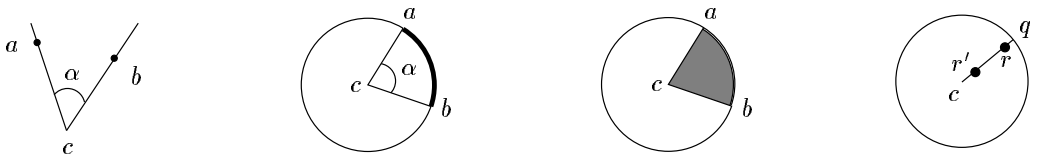
The robots are *fully asynchronous*, i.e., the amount of time spent in each phase of a cycle is finite but otherwise unpredictable. In particular, the robots do not have a common notion of time. As a result, robots can be seen by other robots while moving, and thus computations can be made based on obsolete observations. The robots are *oblivious*, meaning that they do not remember any observations nor computations performed in any previous step. The robots are *anonymous*, meaning that they are a priori indistinguishable by their appearance, and they do not have any kind of identifiers that can be used during the computation. Finally, the robots have *no means of direct communication*: any communication occurs in a totally implicit manner, by observing the other robots' positions.

There are two limiting assumptions concerning *infinity*: **(A1)** The amount of time required by a robot to complete a cycle is not infinite, nor infinitesimally small. **(A2)** The distance traveled by a robot in a cycle is not infinite, nor infinitesimally small (unless it brings the robot to the destination point). As no other assumptions on space exist, the distance traveled by a robot in a cycle is unpredictable.

2.2 The Gathering Problem

The GATHERING PROBLEM is defined as follows.

> Given n robots $r_1, \ldots, r_n$, arbitrarily placed in the plane, with no two robots at the same position, make them gather at one point in a finite number of cycles.



a. b. c. d.

Figure 2: (a) Convex angle $\alpha = \sphericalangle(a, c, b)$. (b) Arc (thick line) and (c) sector (grey part) defined by $\sphericalangle(a, c, b)$. (d) Two robots, r and r', on the same radius.

A relaxed variant of this problem would be to make the robots only move "very close" to each other. This variant is rather easy to solve: each robot computes the center of gravity[2] of all robots, and moves towards it. However, in the GATHERING PROBLEM, we want the robots to meet *exactly* at one point.

For our weak model of robots, the GATHERING PROBLEM is unsolvable, which is shown in [12]:

Theorem 2.1 ([12]). *There exists no deterministic oblivious algorithm that solves the* GATHERING PROBLEM *in a finite number of cycles for a set of $n \geq 3$ robots.*

The proof of the previous theorem relies heavily on the fact that the robots do only know at which points in the plane there is at least one other robot, but they do not know how many robots there are. We say that the robots have the (additional) ability of *multiplicity detection* if they can distinguish whether there is zero, one, or more than one robot at a specific point in the plane. If the multiplicity at a point p is greater than one, we say that there is *strict multiplicity* at p.

Because of Theorem 2.1, in the following we assume that the robots have the ability of *multiplicity detection*. We will heavily exploit multiplicity detection in our algorithms by always gathering the robots at the only position where strict multiplicity occurs.

2.3 Notation

In the rest of the paper, the following notation is used (refer to Figure 2). In general, r indicates any robot in the system (when no ambiguity arises, r is used also to represent the point in the plane occupied by that robot). A *configuration* of the robots at a given time instant t is the set of positions in the plane occupied by the robots at t.

Given two distinct points a and b in the plane, $[a, b)$ denotes the half-line that starts in a and passes through b, and $[a, b]$ denotes the line segment between a and b. Given two half-lines $[c, a)$ and $[c, b)$, we denote by $\sphericalangle(a, c, b)$ the convex angle (i.e., the angle which is at most 180°) centered in c and with sides $[c, a)$ and $[c, b)$. The intersection between the circumference of a circle $\mathcal{C}$ and an angle α at the center of

[2]For n points $p_1, \ldots, p_n$ in the plane, the center of gravity is $c = \frac{1}{n} \sum_{i=1}^{n} p_i$.

$\mathcal{C}$ is denoted by $arc(\alpha)$ (refer to Figure 2.b), and the intersection between α and $\mathcal{C}$ is denoted by $sector(\alpha)$.

Given a circle $\mathcal{C}$ with center c and radius Rad, and a robot r, we will say that r *is on* $\mathcal{C}$ if $dist(rc) = Rad$, where $dist(ab)$ denotes the Euclidean distance between point a and b (i.e., r is on the circumference of $\mathcal{C}$); if $dist(rc) < Rad$, we will say that r *is inside* $\mathcal{C}$. Given two distinct robots r and r', with r inside $\mathcal{C}$, let q be the intersection between the circumference of $\mathcal{C}$ and $[c, r)$. We say that r and r' are on the same radius if $r' \in [c, q]$.

Given points a, b and c, the triangle with these three points as vertices is denoted by $\triangle(a, b, c)$. We use $p \in \triangle(a, b, c)$ to indicate that p is inside the triangle or on its border.

Finally, given a set P, we denote the cardinality of P (i.e., the number of elements in P) by $|P|$.

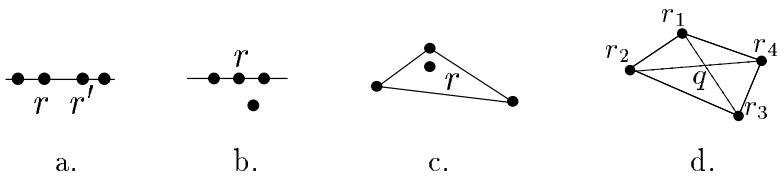
2.4 Weber Points

Given a set of points $P = \{p_1, \ldots, p_n\}$ and a point x in the plane, we define the **Weber distance** between x and P by $WD(x, P) := \sum_{p \in P} dist(p, x)$. A point w is the **Weber point** of point set P if it minimizes the Weber distance between P and any point x in the plane, i.e., if $WD(w, P) = \min_{x \in \mathbb{R}^2} WD(x, P)$. Thus, a Weber point minimizes the sum of all distances to the points in P.

If the points in P are not on a line, then the Weber point always exists, and it is unique. If they are on a line, then the Weber point is unique only for an odd number of points (actually, it is the median point) [14]. Computing the Weber point is possible in many configurations; however, it is not computable in general [4].

3 Algorithms for $n \leq 4$ Robots

In the following, we provide solutions for the GATHERING PROBLEM separately for $n = 2, 3$, and 4 robots. The general idea of the three algorithms is to let the robots reach a configuration where there is exactly one point q in the plane with strict multiplicity (recall that there is no strict multiplicity in the initial configuration, and that the robots can detect multiplicities). When such a configuration is reached, all robots move towards q, avoiding collisions (i.e., q remains the only point with strict multiplicity). This is accomplished by routine `move_to(`q`)` that moves the robot r that calls the routine towards point q in the plane as follows: if r is already at q, it does not move at all; if no robot is on the segment $[r, q]$, then r moves towards q (recall that a robot might not reach its destination point in one step, since it may return to the waiting state before). Otherwise, let r' be the robot on $[r, q]$ closest to r; then r moves towards a point at distance at most $d > 0$ from r'. In this way, no collision occurs except at point q. In our algorithms, we will use `do_nothing()` to indicate that a robot does not want to move at all.



a. b. c. d.

Figure 3: Configurations of 4 points: (a) all on a line; (b) three points on a line; (c) one point inside the convex hull; (d) all on the convex hull.

3.1 Two Robots

The GATHERING PROBLEM is unsolvable for two robots if the robots have no additional abilities [13]. On the other hand, assume that the two robots can detect that they "run into each other" while they are moving, i.e., they can stop immediately when they move on the same line in opposite directions and they meet[3]. Then the problem can be solved easily: each of the two robots simply starts moving towards the other robot, and they stop when they meet.

Result 1. The GATHERING PROBLEM is unsolvable for two robots without additional abilities [13]. If the robots can detect when they "run into each other", there exists an algorithm that solves the GATHERING PROBLEM.

3.2 Three Robots

For $n = 3$ points the Weber point is unique and can be computed [9]. Since it is invariant under straight movement of any of the points towards it [14], we can use it to gather $n = 3$ robots, as shown in the following algorithm.

Algorithm 1 Gathering for $n = 3$

$w :=$ Weber Point of the Three Robots;
`move_to(`w`)`.

Result 2. Three robots that can detect multiplicity can always gather at a point in a finite number of cycles.

3.3 Four Robots

In this section, we present Algorithm 2 that solves the GATHERING PROBLEM for $n = 4$ robots. In particular, the algorithm takes different actions according to the possible initial configurations of the robots, depicted in Figure 3.a–d (these are the only possible configurations for four distinct robots in the plane).

[3]Note that this ability differs from multiplicity detection, since here we assume that they recognize the fact that they "run into each other" *while* they are moving, and that they stop immediately, whereas multiplicity detection is only used during the computation step.

Algorithm 2 Gathering for $n = 4$

If There Is One Point q With Multiplicity > 1 **Then** `move_to(`q`)`.
If All Four Robots Are On A Line **Then**
  r, r' := The Two Median Robots On The Line;
  c := Median Point Of The Two Outer Robots On The Line;
5: **If** I Am r Or r' **Then** `move_to(`c`)` **Else** `do_nothing()`.
If Three Robots Are On A Line **Then**
  r := The Median Robot On The Line;
  `move_to(`r`)`.
CH := Convex Hull Of The Four Robots;
10: **If** One Robot Is Inside CH **Then**
  r := Robot Inside CH;
  `move_to(`r`)`.
If No Robot Is Inside CH **Then**
  q := Intersection Point Of The Two Diagonals Of CH;
15: `move_to(`q`)`.

Observe that in the last case, if one robot reaches point q (computed in Line 14 of the algorithm), we end up in the configuration depicted in Figure 3.b, with q the position of the median robot on the line.

For the correctness of Algorithm 2, let CH be the convex hull of the positions of the four robots at the beginning. If the configuration of the robot is such that

(SM) **There is an unique point q with strict multiplicity**, then all robots move towards q (Line 1). By definition of `move_to()`, and by Assumptions A1 and A2, in a finite number of cycles the gathering is achieved.

Observe that (SM) cannot be an initial configuration, since all robots are at distinct positions at the beginning. In the following, we analyze the behavior of Algorithm 2 in all possible initial configurations.

(A) **The robots are all on the same line**. In this case, only the two median robots on the line, say r and r', are allowed to move towards the median point of the two outer robots on the line, call it c (Lines 3–5). We distinguish the two possible cases.

  (a) Both r and r' reach c simultaneously: Case (SM) above applies.

  (b) Only one robot reaches c, say r (the same argument applies if r' reaches c first). Since the two outermost robots have not moved since the beginning, their median point has not changed. Therefore, Case (A) applies again as long as r' does not reach c. When this happens (by Assumptions A1 and A2, in a finite number of cycles), Case (SM) above applies.

(B) **Three robots are on the same line**. In this case, in Lines 6–8, the algorithm allows to move all robots towards the median robot on the line, say r (therefore,

by definition of `move_to()`, r itself does not move). In a finite number of cycles Case (SM) above applies and the strict multiplicity is at point r. All robots keep moving towards r until the gathering is achieved.

(C) **One robot is inside** CH, say r. In this case, all robots move towards r (Lines 10–12). As in the previous case, in a finite number of cycles Case (SM) above applies with the strict multiplicity at r.

(D) **No robot is inside** CH. In this case, all robots move towards the intersection point of the two diagonals of CH, say q (Lines 13–15). Note that this case applies as long as no robot reaches q. By Assumptions A1 and A2, in a finite number of cycles one or more robots reach q. We distinguish the two possible cases:

  (a) One robot reaches q first: Case (B) above applies, and all robots keep moving towards q.

  (b) More than one robot reach q simultaneously: Case (SM) above applies, and all robots keep moving towards q until the gathering is achieved.

Therefore, we can state the following

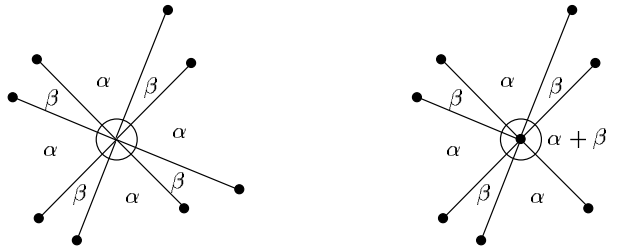
Result 3. Four robots that can detect multiplicity can always gather at a point in a finite number of cycles.

4 Algorithm for $n \geq 5$ Robots in Biangular Configurations

In this section we sketch an algorithm (Algorithm 3) that solves the GATHERING PROBLEM for very specific initial configurations, namely biangular configurations. In Section 5 we will present another algorithm for the GATHERING PROBLEM that works for all initial configurations except totally symmetric ones. As already stated in Section 1, totally symmetric configurations are special cases of biangular configurations, and can thus be solved with the algorithm from this section. However, solving the GATHERING PROBLEM for **all** initial configurations at once remains an open problem that we will discuss in Section 6.

A set of n robots is in *general biangular configuration* if there exist a point c, the *center*, an ordering of the robots, and two angles $\alpha, \beta > 0$ such that each two adjacent robots form an angle α or β w.r.t. c, and the angles alternate (refer to Figure 4.a). The robots are in *degenerated biangular configuration* if there is a robot r, an ordering of the other robots, and two angles $\alpha, \beta > 0$ such that each two adjacent robots form an angle α or β w.r.t. r, and the angles alternate, except for one "gap" where the angle is $\alpha+\beta$ (see Figure 4.b). A general biangular configuration becomes degenerated if one of the robots, namely r, moves to the center c.

If a set of $n \geq 3$ points P is in general or degenerated biangular configuration, then the center of biangularity c is unique, can be computed in finite time, and is



a. b.

Figure 4: (a) General biangular and (b) degenerated biangular configuration of 8 points.

invariant under straight movement of any of the points in its direction; that is, it does not change if any of the points moves towards c. Moreover, c is the Weber point of the points in P.

Algorithm 3 Gathering for $n \geq 5$ robots in Biangular Configuration

If There Is One Point q With Multiplicity > 1 **Then** `move_to(`q`)`.
If The Robots Are In Degenerated Biangular Configuration **Then**
    c := Center Of Degenerated Biangularity;
    `move_to(`c`)`.
5: **Else** %The Robots Are In General Biangular Configuration%
    c := Center Of Biangularity;
    `move_to(`c`)`.

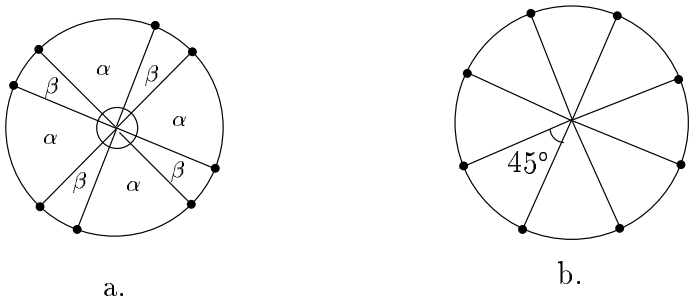
For the correctness of Algorithm 3, note that there are only two possible initial configurations of the robots: either they are in general or in degenerated biangular configuration.

In the first case, the algorithm moves all robots straight towards c (Lines 5–7). As long as none of the robots reaches c, the configuration remains general biangular; hence the algorithm moves all of them towards c. By Assumptions A1 and A2, in a finite number of cycles, one or more robots reach c. We analyze separately the two cases.

1. If more than one robot reach c simultaneously, then c is the unique point in the plane with strict multiplicity. In this case, all the robots keep moving towards c (Line 1), until gathering is achieved.

2. If only one robot reaches c first, then the configuration becomes degenerated with center c[4], and the algorithm runs into Lines 2–4, where all robots are still allowed to move towards c. Therefore, in a finite number of cycles, at least two robots are on c, and previous case applies.

[4]If a general biangular configuration with center c turns into a degenerated biangular configuration because one of the robots reaches c, then the center of the degenerated biangular configuration is again c.



a. b.

Figure 5: Totally symmetric configurations.

If the robots are in degenerated biangular configuration at the beginning, the algorithm moves the robots towards the center of biangularity. Again, after a finite number of cycles, one or more robots reach the center, and a point with strict multiplicity is obtained. Therefore, we have the following

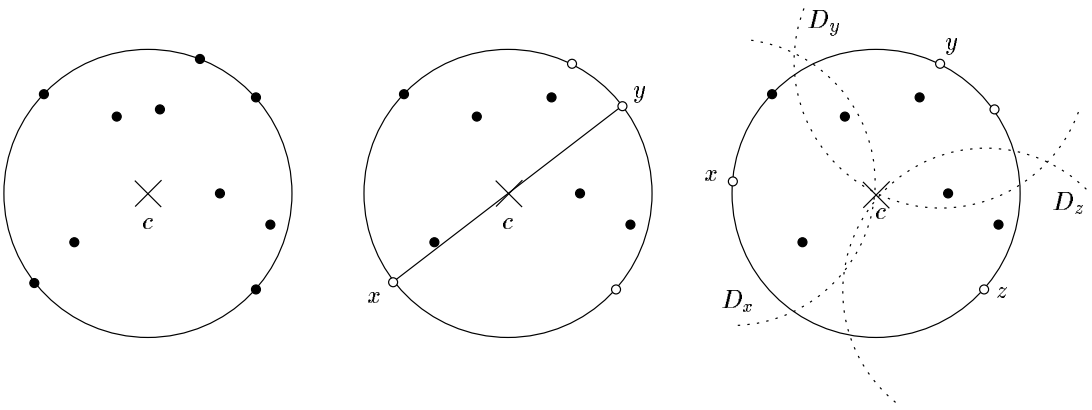
Result 4. More than four robots that can detect multiplicity can always gather at a point in a finite number of cycles if the initial configuration is biangular.

5 Algorithm for $n \geq 5$ Robots in Non Totally Symmetric Configurations

We will now present an algorithm that solves the GATHERING PROBLEM for almost all initial configurations of the robots except those that are *totally symmetric*. Here, a configuration is totally symmetric if it is biangular with some center c, and all robots are on a circle with center c (see Figure 5). Totally symmetric configurations are covered by the algorithm for biangular configurations that we presented in the previous section.

If the robots are not in a totally symmetric configuration at the beginning, the main idea of the algorithm is to let the robots reach a configuration where there is only one point in the plane with strict multiplicity. Then, all robots will gather there. More specifically, first the smallest circle enclosing all robots is computed. Such a circle always exists and is unique (see Section 5.1 below). Then a subset of the robots is chosen to move towards the center of this circle. As soon as two or more robots reach the center, it is a point with strict multiplicity (the only one), and all robots gather in this point. The algorithm guarantees the following invariants:

- the subset of the robots elected to move is chosen such that the smallest enclosing circle of all robots does not change while these robots approach the center of the circle;
- all movements of the robots are mainly straight towards the center of the circle (except for one case), until a point with strict multiplicity occurs;
- there is at most one point with strict multiplicity.



a. b. c.

Figure 6: (a) The smallest enclosing circle of 10 points on the plane. (b) Lemma 5.1, with x, y, and c on a line. The white points belong to $S \subseteq P$. (c) Lemma 5.1, with no two points in S on a line with c.

Before introducing the main algorithm, we first introduce more formally some definitions that will be used in the following, namely the smallest circle enclosing n points (*smallest enclosing circle*), and the *string of angles*.

5.1 Smallest Enclosing Circles

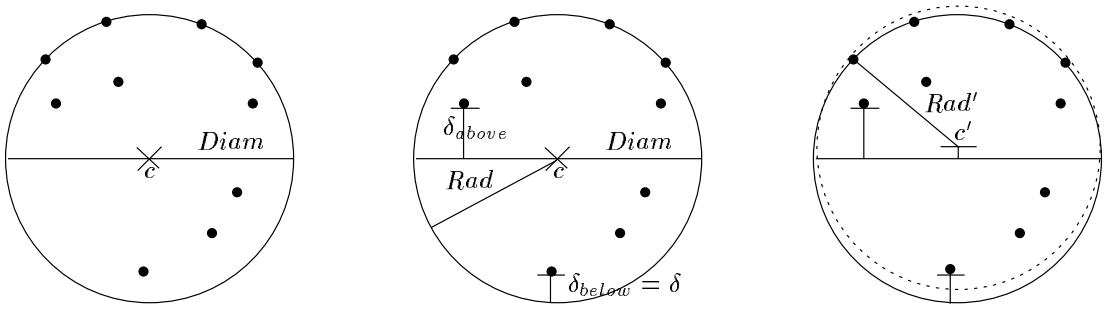
Given a set of n distinct points P in the plane, the *Smallest Enclosing Circle* of the points is the smallest circle enclosing them; in other words, it is the circle with minimum radius such that all points from P are inside or on the circle (see Figure 6.a). We denote it by $SEC(P)$, or SEC if set P is unambiguous from the context. The smallest enclosing circle of a set of n points is unique and can be computed in $O(n \log n)$ time [15].

Obviously, the smallest enclosing circle of P remains invariant if we remove all or some of the points from P that are inside $SEC(P)$. The following lemma shows that we can even remove some points from P that are **on** $SEC(P)$ without changing the smallest enclosing circle:

Lemma 5.1. *Given a set of n points P and a subset $S \subseteq P$, let c be the center of $SEC(P)$. If all points in S are on $SEC(P)$ and c is inside the convex hull of the points in S, then $SEC(P) = SEC(S)$.*

Proof. Let c be the center and Rad be the radius of $SEC(P)$, and let c' be the center of $SEC(S)$, and Rad' be its radius. Since $S \subseteq P$, we have $Rad' \leq Rad$.

If there are two points $x, y \in S$ such that x, y, and c are on a line, then $dist(x, y) = 2 \cdot Rad$ (see Figure 6.b). This implies that the diameter of $SEC(S)$ is at least $2 \cdot Rad$, which yields $Rad' \geq Rad$; hence, $Rad' = Rad$. Since $SEC(P)$ encloses all points from S and has radius Rad, and since the smallest enclosing circle is unique, we have $SEC(S) = SEC(P)$.



a. b. c.

Figure 7: Lemma 5.2. (a) All points in P that are on $\mathcal{C}$ are inside one of the two half-circles delimited by $Diam$. (b) Distances δ_{above} and δ_{below}. (c) The dotted circle is the circle $\mathcal{C}_{shifted}$ centered in c' and having radius $Rad' = \max_{p \in P} dist(c', p)$.

If no two points from S are on a line with c, since c is inside the convex hull of the points in S, there exist three points $x, y, z \in S$ such that c is inside[5] $\triangle(x, y, z)$ (see Figure 6.c). Let D_x, D_y and D_z be the disks with radius Rad and center x, y, and z, respectively. Since x, y, and z have distance Rad from c, these three disks intersect in exactly one point, namely c. On the other hand, each of the three points x, y and z has at most distance Rad' from c', the center of $SEC(S)$. Thus, since $Rad' \leq Rad$, c' has to be in the intersection of all the three disks. Therefore, $c' = c$, and $SEC(S) = SEC(P)$. □

Let $Diam$ be a diameter of $SEC(P)$. We call the *sides* of $Diam$ the two half-circles delimited by $Diam$. Moreover, we say that a point is strictly on one side of $Diam$ if it is not on $Diam$. The following lemma locates the possible areas on $SEC(P)$ where the points in P can be.

Lemma 5.2. *Given a set of n points P and their smallest enclosing circle $SEC(P)$, there exists no diameter $Diam$ of $SEC(P)$ such that all points in P are strictly on one side of $Diam$.*

Proof. Let c be the center and Rad be the radius of $\mathcal{C} = SEC(P)$. By contradiction, let us assume that there exists a diameter $Diam$ of $\mathcal{C}$ such that all points in P that are on $\mathcal{C}$ are all strictly inside one side of $Diam$; moreover, let us assume without loss of generality that $Diam$ is horizontal (see Figure 7.a). We will show how to find a point $c' \neq c$ and a radius $Rad' < Rad$ such that all points in P are inside the circle with center c' and radius Rad', having a contradiction since $\mathcal{C}$ was assumed to be the smallest enclosing circle of the points in P.

Let δ_{above} be the minimal vertical distance from diameter $Diam$ to any point from P that is strictly above $Diam$, δ_{below} be the minimal vertical distance from the circumference $\mathcal{C}$ to any point from P that is below or on $Diam$, and $\delta = \min\{\delta_{above}, \delta_{below}\}$;

[5] In fact, connecting one vertex of the convex hull to all other vertices, c is clearly inside one of the triangles determined in this way.

hence, $\delta > 0$ (see Figure 7.b). Let c' be the point on the vertical line through c, and above c such that $dist(c, c') = \frac{\delta}{2}$ (see Figure 7.c). Then, by definition of δ, all points in P are all inside the circle $\mathcal{C}_{shifted}$ that has center in c' and radius Rad. Moreover, observe that (I) no point from P is on $\mathcal{C}_{shifted}$.

Let $Rad' = \max_{p \in P} dist(c', p)$. Then, all points in P are inside or on the circle $\mathcal{C}'$ with center c' and radius Rad'; furthermore, by above observation (I), $Rad' < Rad$. Thus, circle $\mathcal{C}$ is not the smallest enclosing circle for the points in P, having a contradiction. □

In the following lemma, we show how to construct a subset S of P such that $SEC(S) = SEC(P)$.

Lemma 5.3. *Given a set P of n points, there exists a subset $S \subseteq P$ such that $|S| \leq 3$ and $SEC(S) = SEC(P)$.*

Proof. If $|P| \leq 3$, the lemma trivially follows. Otherwise, let c be the center of $SEC(P)$. First, observe that c is inside or on the convex hull of the points in P that are on $SEC(P)$; otherwise, all these points would lie all on one half of the circumference of $SEC(P)$ (see Figure 7.a), and, by Lemma 5.2, $SEC(P)$ would not be the smallest circle that encloses the points in P.

If there exist two distinct points $x, y \in P$ that are on $SEC(P)$ such that x, y and c are on a line, then, by Lemma 5.1, the lemma follows with $S = \{x, y\}$. Otherwise, by above observation, there exist three points $x, y, z \in P$ that are on $SEC(P)$ such that c is inside $\triangle(x, y, z)$. Then, by Lemma 5.1, the lemma follows with $S = \{x, y, z\}$. □

5.2 String of Angles

Given n distinct points $p_1, \ldots, p_n$ in the plane, let SEC be the smallest enclosing circle of the points, and c be its center. For an arbitrary point p_k, $1 \leq k \leq n$, we define the *string of angles* $SA(p_k)$ by the following algorithm (refer to Figure 8):

`Compute_SA(`p_k`)`
  $p := p_k, i := 1$;
  While $i \neq n + 1$ **Do**
    $p' :=$ `Succ(`p`)`;
    $SA[i] := \sphericalangle(p, c, p')$;
    $p := p'$; $i := i + 1$;
  End While
  Return SA.

Here, all angles are oriented clockwise (note that the robots do not have a common knowledge on orientations; however, each single robot can distinguish between a "local" clockwise and counterclockwise orientation). The successor of p, computed by `Succ(`p`)`, is (refer to Figure 9)

- either the point $p_i \neq p$ on $[c, p)$, such that $dist(c, p_i)$ is minimal among all points $p_j \neq p$ on $[c, p)$ with $dist(c, p_j) > dist(c, p)$, if such a point exists; or

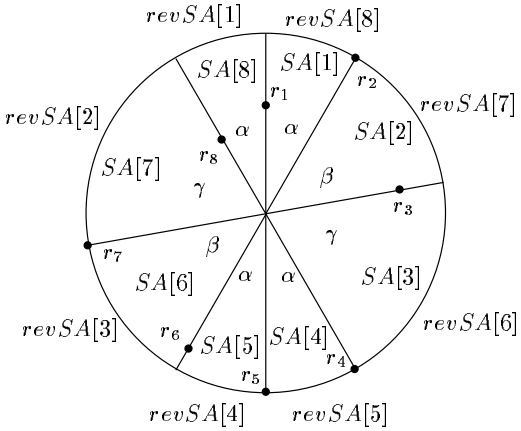


Figure 8: Example of the string of angles computed by `Compute_SA(`r_1`)`, with a clockwise orientation of SEC. With $\alpha = 25°, \beta = 60°$, and $\gamma = 70°$, we have $SA(r_1) = \langle \alpha, \beta, \gamma, \alpha, \alpha, \beta, \gamma, \alpha \rangle = \langle 25°, 60°, 70°, 25°, 25°, 60°, 70°, 25° \rangle$; $LexMinString = \langle \alpha, \alpha, \beta, \gamma, \alpha, \alpha, \beta, \gamma \rangle$; $StartSet = \{4, 8\}$, and $revStartSet = \emptyset$.

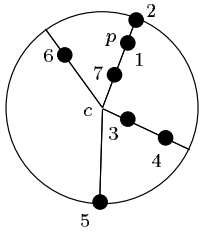


Figure 9: Routine `Succ(`p`)` in `Compute_SA()`. Orientation is clockwise. The points are numbered according to routine `Succ()`; that is `Succ(`p`)`=2, `Succ(2)`=3, and so on.

- the point $p_i \neq p$ such that there is no other point inside $sector(\sphericalangle(p, c, p_i))$, and there is no other point on the line segment $[c, p_i]$.

Instead of $SA(p_k)$, we write SA if we do not consider a specific point p_k. Given p_k, procedure `Succ()` defines a unique successor, and thus a unique string of angles. Given two starting points p_k and p_ℓ, then $SA(p_k)$ is a cyclic shift of $SA(p_\ell)$. Given an angle α in SA, then we can associate it with its defining point; i.e., if $\alpha = \sphericalangle(p, c, p')$, then we say that α is *associated* to p: we will write $p = \mathfrak{r}(\alpha)$. Alternatively, since α is stored in SA, say at position i (i.e., $SA[i] = \alpha$), we will denote the point associated to α also by $\mathfrak{r}(i)$, saying that $\mathfrak{r}(i)$ is the point associated to position i in SA. In the example depicted in Figure 8, $\mathfrak{r}(SA[3]) = \mathfrak{r}(\gamma) = \mathfrak{r}(3) = r_3$.

We say that SA is *general* if it does not contain any zeros; otherwise, at least two points are on a line starting in c (a radius), and we call the string of angles *degenerated*.

We define the *reverse string of angles* $revSA$ in an analogous way: it is the string of angles according to a counterclockwise orientation (i.e., $revSA$ is the reverse of

SA).

Given two strings $s = s_1, \ldots, s_n$ and $t = t_1, \ldots, t_n$, we say that s is *lexicographically smaller* than t if there exists an index $k \in \{1, \ldots, n\}$ such that $s_i = t_i$ for all $1 \leq i < k$, and $s_k < t_k$. We write $s <_{lex} t$. Let $LexMinString$ be the lexicographically minimal string among all strings of angles (in both orientations), i.e., $LexMinString := min(\{SA(p_i) \mid 1 \leq i \leq n\} \cup \{revSA(p_i) \mid 1 \leq i \leq n\})$. Let $StartSet$ be the set of all indices in SA where $LexMinString$ starts, i.e., $StartSet = \{i \mid 1 \leq i \leq n, SA(p_i) = LexMinString\}$, and let $revStartSet$ be the set of all indices in $revSA$ where $LexMinString$ starts.

5.3 The Main Algorithm

In Algorithm 4, we present the main part of the algorithm that solves the GATHERING PROBLEM for $n \geq 5$ robots if the initial configuration is not totally symmetric. The algorithm guarantees that the smallest enclosing circle SEC of the robots does not change as long as there is no point with strict multiplicity. Moreover, the string of angles SA does not change until a point with strict multiplicity occurs, or SA becomes degenerated (note that, if a single robot reaches the center of SEC, then SA is degenerated). Since the initial configuration is not totally symmetric, a strict subset of the robots can be elected to move inside SEC. During the movement, the occurrence of any totally symmetric configuration is avoided by the algorithm. The three subroutines in Lines 17–19 are presented and discussed separately in the following sections. Recall that `move_to(`p`)` means that a robot moves towards p, but only if no other robot bars the way (cf. Section 3).

We will now proof the correctness of Algorithm 4. If the configuration of the robots is such that

(SM) **There is an unique point q with strict multiplicity**, then all robots move towards q (Line 1). By definition of routine `move_to()`, any collision is avoided, and in a finite number of cycles the gathering is achieved in q.

Observe that (SM) cannot be an initial configuration, since the robots are at distinct positions at the beginning. In the following, we consider the run of the algorithm for all possible initial configurations of the robots (refer to Figure 10). In particular, the following scenarios can occur at the beginning.

(A) **All robots are on SEC.** Lines 15–17 are executed. The corresponding sub cases are discussed in the following sections.

(B) **All robots except one, say $\tilde{r}$, are on SEC, $\tilde{r}$ is not at c, and $\tilde{r}$ is on the same radius as some robot** s. In this case, SA is degenerated, and the algorithm performs Line 21, with $r' = s$, where only $\tilde{r}$ is allowed to move towards s. If it does not reach s in one cycle, then $\tilde{r}$ moves closer to s and remains on the same radius; hence, the same case applies again. By Assumptions A1 and A2, in a finite number of cycles r reaches s, and a configuration with a (unique) point with strict multiplicity is reached, and Case (SM) above applies.

Algorithm 4 Gathering for $n \geq 5$ (initial configuration not totally symmetric)

If There Is One Point q With Multiplicity > 1 **Then** `move_to(`q`)`.
SEC := Smallest Enclosing Circle Of All Robots;
c := Center Of SEC;
If One Robot r Is At Point c **Then**
5: **If** No Other Robot Is Inside SEC **Then**
q := Position Of An Arbitrary Robot On SEC;
If I Am r **Then** `move_to(`q`)` **Else** `do_nothing()`.
Else % At Least On Other Robot Is Inside SEC Besides r %
If I Am Inside SEC **Then** `move_to(`c`)` **Else** `do_nothing()`.
10: **Else** % No Robot Is At c %
r := An Arbitrary Robot;
SA := `Compute_SA(`r`)`; %String Of Angles w.r.t. r%
If SA Is General **Then**
$StartSet, revStartSet$:= Indices Where Lex. Minimal String Starts;
15: **If** $|StartSet \cup revStartSet| = 1$ **Then** `OneStartingIndex()`.
If $|StartSet \cup revStartSet| = 2$ **Then** `TwoStartingIndices()`.
If $|StartSet \cup revStartSet| > 2$ **Then** `ManyStartingIndices()`.
Else %SA Is Degenerated%
If Only One Robot $\tilde{r}$ Is Inside SEC **Then**
20: r' := Robot On SEC On Same Radius as $\tilde{r}$;
If I Am $\tilde{r}$ **Then** `move_to(`r'`)` **Else** `do_nothing()`.
Else % More Than One Robot Is Inside SEC %
If I Am Inside SEC **Then** `move_to(`c`)`.
Else `do_nothing()`.

(C) **All robots except one, say r, are on SEC, and r is at c.** The algorithm runs into Lines 5–7, where only r is allowed to move towards an arbitrary robot q that is on SEC. It either reaches q in one cycle, obtaining a (unique) point with strict multiplicity, and case (SM) above applies; or r stops during the movement towards q. In the second case, r is the only robot inside SEC, and it is on the same radius as robot q; hence, SA becomes degenerated and the argument follows as in previous Case (B).

(D) **All robots except one, say $\tilde{r}$, are on SEC, $\tilde{r}$ is not at c, and $\tilde{r}$ is not on the same radius as any other robot.** In this case SA is general, and the algorithm runs into Lines 15–17. These cases are discussed below.

(E) **More than one robot is inside SEC, and one robot r is at c.** The algorithm executes Lines 8–9. All robots that are inside SEC will move towards c, while r, already at c, does not move at all. The robots inside SEC move according to routine `move_to()`, avoiding collisions. If no robot reaches c in one cycle, then the robots inside SEC move closer towards c, and the same case applies again. By Assumptions A1 and A2, in a finite number of cycles at least one robot reaches c, and a configuration where c is the (unique) point

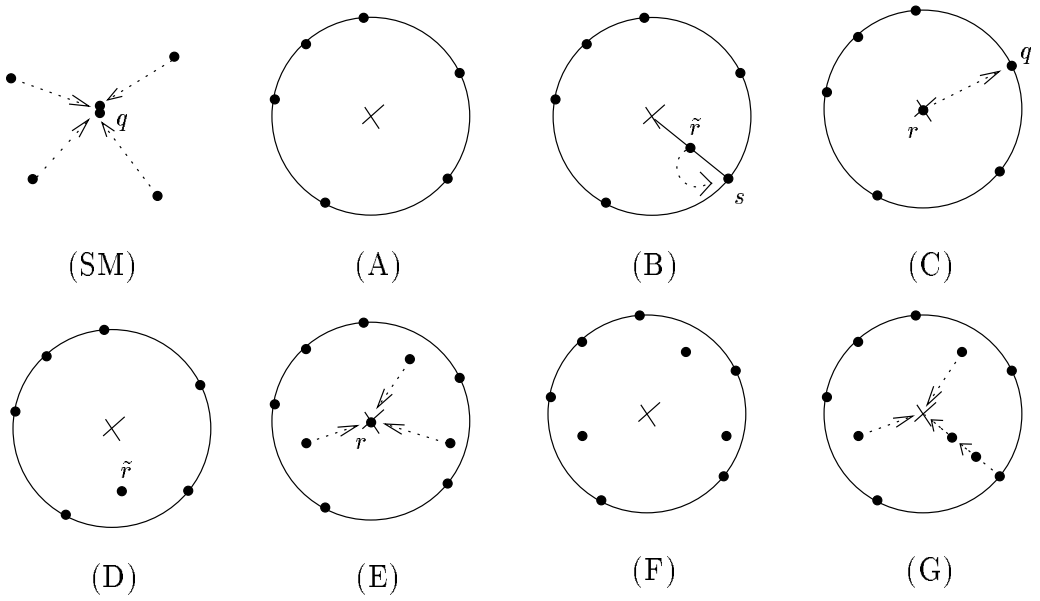


Figure 10: Algorithm 4: (SM) special case with one point with strict multiplicity; (A)–(G) the possible initial configuration of the robots.

with strict multiplicity is reached; hence, and Case (SM) above applies.

(F) **More than one robot is inside *SEC*, no robot r is at c, and no two robots are on the same radius**. Again, the algorithm runs into Lines 15–17, which are discussed below.

(G) **More than one robot is inside *SEC*, no robot r is at c, and at least two robots are on the same radius**. *SA* is degenerated, and the algorithm runs into Lines 22–24: all robots that are inside *SEC* will move towards c by calling routine `move_to(`c`)`, hence avoiding collisions. By Assumptions A1 and A2, in a finite number of cycles one or more robots reach c. If only one robot reaches c first, then the algorithm runs into Lines 8–9, and case (E) above applies. If more than one robot reach c at the same time, we get a (unique) strict multiplicity in c, and Case (SM) above applies.

Therefore, we can state the following

Theorem 5.1. *If the initial configuration of the robots is such that Cases (B), (C), (E), or (G) above apply, then in a finite number of cycles the robots gather at a point.*

In the following sections, we analyze what happens in Cases (A), (D), and (F); in these cases, the initial configuration is such that either subroutine `OneStartingIndex()`, or `TwoStartingIndices()`, or `ManyStartingIndices()` is called in Lines 15–17 of the main algorithm.

5.4 Subroutine OneStartingIndex

In this section, we describe subroutine `OneStartingIndices()` (see SubRoutine 1), called in Line 15 of the main algorithm. This case is executed when at the beginning SA is general and $|StartSet \cup revStartSet| = 1$.

Let $StartSet \cup revStartSet = \{x\}$ and $SA(x) = \alpha_1, \ldots, \alpha_n$; then $revSA(x) = \alpha_n, \ldots, \alpha_1$. The following lemma shows that either $SA(x)$ or $revSA(x)$ can be lexicographically minimal, but not both of them.

Lemma 5.4. *If $StartSet \cup revStartSet = \{x\}$, then either $SA(x) = LexMinString$ or $revSA(x) = LexMinString$.*

Proof. By contradiction, let us assume that $SA(x) = revSA(x) = LexMinString$. Then $\alpha_n = \alpha_1$. Since x is the only starting position of $LexMinString$, we have $SA(x) = \alpha_1, \ldots, \alpha_n <_{lex} \alpha_n, \alpha_1, \ldots, \alpha_{n-1}$. Furthermore, since $\alpha_n = \alpha_1$, it follows that $\alpha_2, \ldots, \alpha_n <_{lex} \alpha_1, \ldots, \alpha_{n-1}$, and thus $\alpha_2, \ldots, \alpha_n, \alpha_1 <_{lex} \alpha_1, \ldots, \alpha_{n-1}, \alpha_n = SA(x)$, in contradiction to the assumption that $SA(x)$ is lexicographically minimal. □

In the following, we will assume that $SA(x) = LexMinString$ (the case $revSA(x) = LexMinString$ can be handled similarly). This gives us an unique ordering of the robots, starting in x. For the case when all robots are on SEC, we define a routine `Elect()` that elects a unique robot as follows: it chooses the first robot r – according to the ordering defined before – such that the smallest enclosing circle does not change by removing r from the set of robots. Such a robot exists due to Lemma 5.3, since $n \geq 5$. Using this routine, Subroutine `OneStartingIndex()` is defined as follows:

SubRoutine 1 OneStartingIndex

If All Robots Are On SEC **Then**
  r := `Elect()`;
  If I Am r **Then** `move_to(`c`)` **Else** `do_nothing()`.
If Only One Robot r' Is Inside SEC **Then**
5: **If** I Am r' **Then** `move_to(`c`)` **Else** `do_nothing()`.
If More Than One Robot Is Inside SEC **Then**
  If I Am Inside SEC **Then** `move_to(`c`)` **Else** `do_nothing()`.

To prove that gathering is achieved when at the beginning the configuration of the robots is such that this subroutine is called (i.e., in cases (A), (D), and (F) of Section 5.3), first note that,

Note 1. As long as Subroutine 1 is executed, then SA remains general. In particular, all movements of the robots are straight towards c; thus, the string of angles does not change and SEC remains invariant during the movements, until some robot reaches c.

In the following, we analyze the possible initial configurations that make Subroutine 1 to be called.

(1) **All robots are on** *SEC*. A unique robot r is elected by routine `Elect()`, and it moves towards c; all other robots are not allowed to move, and *SEC* remains invariant. If r stops on $[rc]$ before reaching c, then Case (2) applies (Lines 5 of the subroutine is performed), and r keeps moving towards c. By Assumptions A1 and A2, in a finite number of cycles, r reaches c. As soon as this happens, Lines 5–7 of the main algorithm are executed, and case (C) applies.

(2) **Only one robot r' is inside** *SEC*. Since SA is general when Subroutine 1 is called, r' is not on the same radius as any other robot. Then r' moves towards c (Line 5 of the subroutine). If r' does not reach c in one cycle, then this case applies again. By Assumptions A1 and A2, robot r' reaches c in a finite number of cycles. At this point, Lines 5–7 of the main algorithm are executed, and case (C) of the main algorithm applies. We note that this case holds if, in the initial configuration, r was either inside *SEC*, or on *SEC* and elected in the initial configuration due to routine `Elect()`.

(3) **More than one robot is inside** *SEC*. All robots that are inside *SEC* will move towards c. Since SA is general, there are no two robots on the same radius, and Lines 6–7 of Subroutine 1 are executed. If no robot reaches c, then this case applies again. By Assumptions A1 and A2, at least one robot will reach c in a finite number of cycles. If only one robot reaches c first, Lines 8–9 of the main algorithm are executed, and Case (E) applies. Otherwise (more than one robot reaches c simultaneously), c is the (unique) point with strict multiplicity, and Case (SM) of the main algorithm applies.

Therefore, we can state the following

Theorem 5.2. *If the initial configuration of the robots is such that Subroutine 1 is executed, then in a finite number of cycles the robots gather at a point.*

5.5 Subroutine TwoStartingIndices

In this section, we describe subroutine `TwoStartingIndices()` (see SubRoutine 2), called in Line 16 of the main algorithm. This case is executed when at the beginning SA is general and $StartSet \cup revStartSet = \{x, y\}$.

In the following lemma we show that, if there are two different starting positions x, y of $LexMinString$, then $LexMinString$ starts in each of the positions in only one direction, either clockwise or counterclockwise.

Lemma 5.5. *If $StartSet \cup revStartSet = \{x, y\}$, then it is not possible that $SA(x) = revSA(x) = LexMinString$ or $SA(y) = revSA(y) = LexMinString$.*

Proof. Similarly to the proof of Lemma 5.4, let us assume by contradiction that $SA(x) = revSA(x) = LexMinString$, and let $SA(x) = \alpha_1, \ldots, \alpha_n$. We have $revSA(x) = \alpha_n, \ldots, \alpha_1$, and $\alpha_n = \alpha_1$. This yields $SA(x) = \alpha_1, \ldots, \alpha_n \leq_{lex} \alpha_n, \alpha_1, \ldots, \alpha_{n-1}$. Since $\alpha_n = \alpha_1$, then $\alpha_2, \ldots, \alpha_n \leq_{lex} \alpha_1, \ldots, \alpha_{n-1}$, and thus $\alpha_2, \ldots, \alpha_n, \alpha_1 \leq_{lex} \alpha_1, \ldots, \alpha_{n-1}, \alpha_n = SA(x)$. Since $SA(x)$ is lexicographically minimal, $\alpha_2, \ldots, \alpha_n, \alpha_1$

must be lexicographically minimal as well, and we have $SA(x) = \alpha_1, \ldots, \alpha_n = \alpha_2, \ldots, \alpha_n, \alpha_1$. This implies immediately $\alpha_i = \alpha_{i+1}$, for all $1 \leq i \leq n-1$. Therefore, $\alpha_i = \alpha_1$ for all $2 \leq i \leq n$; that is, $SA(x) = \alpha_1, \alpha_1, \ldots, \alpha_1$. Thus, $LexMinString$ starts in SA in every position $i \in \{1, \ldots, n\}$, in contradiction to the assumption that there are at most two starting positions of $LexMinString$ in SA. □

Since $LexMinString$ can not start at the same position in opposite directions, there are only two cases: either x and y are in the same set ($StartSet$ or $revStartSet$), or they are in different sets. Recall that $\mathfrak{r}(x)$ is the robot associated with index x.

Lemma 5.6. *If x and y are both in $StartSet$ or both in $revStartSet$, then $\sphericalangle(\mathfrak{r}(x), c, \mathfrak{r}(y)) = 180°$.*

Proof. Without loss of generality, let us assume that $x, y \in StartSet$. Let $SA(x) = \alpha_1, \ldots, \alpha_n$. Then $SA(y) = \alpha_{k+1}, \ldots, \alpha_n, \alpha_1, \ldots, \alpha_k$ for some $k \in \{2, \ldots, n-1\}$, and we have $SA(x) = SA(y) = LexMinString$. We say that the starting positions of $LexMinString$ are 0 and k. Then, for all $1 \leq i \leq n$ we have

$$\alpha_i = \alpha_{k+i} \tag{1}$$

(here, we abuse notation and identify any index $\ell > n$ with index $\ell - n$; e.g. $\alpha_{n+3} = \alpha_3$). If $k < \lceil \frac{n}{2} \rceil$, then there would be a third starting point of $LexMinString$ at position $2k < n$, which contradicts the hypothesis that $|StartSet| = 2$. Thus, $\lceil \frac{n}{2} \rceil \leq k < n$. Let us assume $k > \lceil \frac{n}{2} \rceil$, and let us consider the string $s = \alpha_{n-k+1}, \ldots, \alpha_n, \alpha_1, \ldots, \alpha_{n-k}$. By Equation (1), it follows

$$\begin{array}{rcccl}
\alpha_{n-k+1} & = & \alpha_{(n-k+1)+k} & = & \alpha_1 \\
\alpha_{n-k+2} & = & \alpha_{(n-k+2)+k} & = & \alpha_2 \\
 & \vdots & & & \\
\alpha_n & = & \ldots & = & \alpha_k \\
\alpha_1 & = & \ldots & = & \alpha_{k+1} \\
 & \vdots & & & \\
\alpha_{n-k} & = & \ldots & = & \alpha_n,
\end{array}$$

hence $s = LexMinString$. Since by hypothesis $k > \lceil n/2 \rceil$, we have $n - k + 1 < \lfloor \frac{n}{2} \rfloor + 1$, hence

$$n - k + 1 \leq \lceil \frac{n}{2} \rceil. \tag{2}$$

Equation (2) and $\lceil \frac{n}{2} \rceil < k < n$ imply that $n - k \neq k$ and $n - k \neq 0$. Therefore, $n - k$ is a (distinct) third starting position for $LexMinString$, having a contradiction.

Thus, $k = \frac{n}{2}$, and 0 and $\frac{n}{2}$ are the starting positions of $LexMinString$. Then $360° = \sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{\frac{n}{2}} \alpha_i + \sum_{i=\frac{n}{2}+1}^{n} \alpha_i = \sum_{i=1}^{\frac{n}{2}} \alpha_i + \sum_{i=1}^{\frac{n}{2}} \alpha_{i+k} = 2 \cdot \sum_{i=1}^{\frac{n}{2}} \alpha_i = 2\sphericalangle(\mathfrak{r}(x), c, \mathfrak{r}(y))$. Thus, the angle between $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$ w.r.t. c is 180°. □

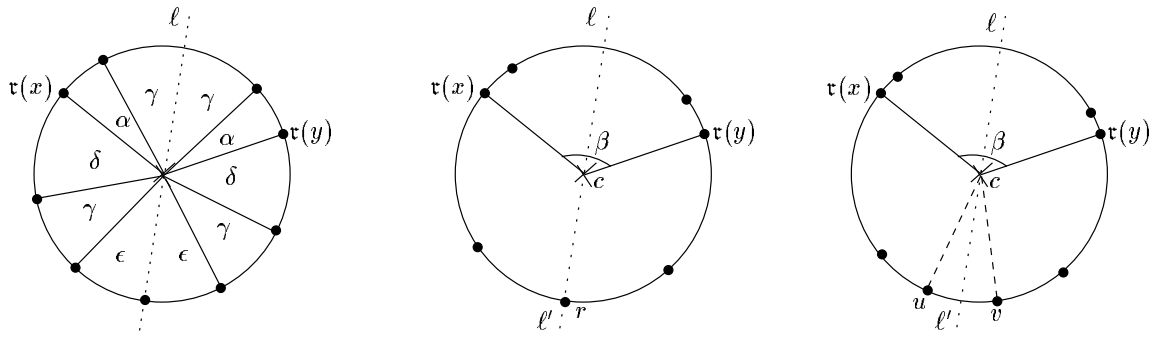


Figure 11: (a) The line ℓ that runs through c and bisects $\beta = \sphericalangle(\mathfrak{r}(x), c, \mathfrak{r}(y)) = 2\alpha+2\gamma$ is a symmetry axis for the angles that the robots form w.r.t. c. In the depicted example, $x \in StartSet$, $y \in revStartSet$, and $SA(x) = revSA(y) = LexMinString = \langle \alpha, \gamma, \gamma, \alpha, \delta, \gamma, \epsilon, \epsilon, \gamma, \delta \rangle$. (b) One robot r opposite to $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$. (c) Two robots u and v opposite to $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$.

It follows by Lemma 5.6 that, since $k = \frac{n}{2}$, there can only be two starting positions with the same orientation if n is even. Moreover, the above lemma suggests us which robot to move when $x, y \in StartSet$ (or $x, y \in revStartSet$) and all robots are on SEC: since $n \geq 5$, it is sufficient to not move $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$, while all other robots can move towards c. In this way, in fact, SEC remains invariant, since $\sphericalangle(\mathfrak{r}(x), c, \mathfrak{r}(y)) = 180°$ (by Lemma 5.1).

Assume now that x and y do not belong to the same set ($StartSet$ or $revStartSet$). Without loss of generality, let us assume that $x \in StartSet$ and $y \in revStartSet$, and let $\beta = \sphericalangle(\mathfrak{r}(x), c, \mathfrak{r}(y))$. Moreover, let all robots be on SEC. If $\beta = 180°$, similarly to what observed for Lemma 5.6, we can move all robots except x and y without changing SEC.

For the case $\beta < 180°$, we define the *opposite robots* of x and y as follows (see Figure 11): let ℓ be the line that runs through c and that bisects β. Since $x \in StartSet$ and $y \in revStartSet$, and $SA(x) = revSA(y) = LexMinString$, ℓ is a symmetry axis for the angles that the robots form w.r.t. c (as long as all robots are on SEC, ℓ is a symmetry axis for the robots' positions as well; see the example depicted in Figure 11.a). We choose one or two robots that are not inside the sector defined by β as follows: let ℓ' be the half-line of ℓ that starts in c and that does not bisect β. If there is a robot r on ℓ', then r is the opposite robot of $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$ (see Figure 11.b). Otherwise, we choose as opposite robots the two robots u and v such that ℓ' bisects $\sphericalangle(u, c, v)$ and such that $\sphericalangle(u, c, v)$ is minimal (see Figure 11.c; note that, if $\beta = 180°$, there are no opposite robots). Observe that the construction of the opposite robots guarantees that c is inside the convex hull of $\mathfrak{r}(x)$, $\mathfrak{r}(y)$, and their opposite robot(s). In the algorithm, we will move all robots except these three (resp. four) robots towards c. This guarantees that SEC does not change (by Lemma 5.1).

In the following Subroutine 2, we use a function `move_into_SEC()` that works as follows: if a robot is on SEC, then it slightly moves to the inside of SEC, say half of

its distance to c. If the robot is already inside SEC, it does not move at all. Using this, we can ensure that a robot does not reach c (and thus switch case in the main algorithm) unless at least one other robot is already inside SEC. Furthermore, note that we handle the cases $StartSet = \emptyset$ and $revStartSet = \emptyset$ simultaneously, since they are totally equivalent.

To prove that gathering is achieved when at the beginning the configuration of the robots is such that this subroutine is called (i.e., in cases (A), (D), and (F) of Section 5.3), we first observe that

SubRoutine 2 TwoStartingIndices

$\{x, y\} := StartSet \cup revStartSet$;
If All Robots Are On SEC **Then**
  If $revStartSet = \emptyset$ (resp. $StartSet = \emptyset$) **Then**
    If I Am Not $\mathfrak{r}(x)$ **And** I Am Not $\mathfrak{r}(y)$ **Then** `move_into_SEC()`
    Else `do_nothing()`.
  Else $\%_{|StartSet| = |revStartSet| = 1}\%$
    $Opposite = \{$Robot(s) Opposite To $\mathfrak{r}(x)$ And $\mathfrak{r}(y)$ $\}$;
    $Stationary = \{\mathfrak{r}(x), \mathfrak{r}(y)\} \cup Opposite$;
    If I Am Not In $Stationary$ **Then** `move_into_SEC()`
    Else `do_nothing()`.
If Only One Robot r Is Inside SEC **Then**
  If $revStartSet = \emptyset$ (resp. $StartSet = \emptyset$) **Then**
    If $r = \mathfrak{r}(x)$ **Or** $r = \mathfrak{r}(y)$ **Then**
      If I Am r **Then** `move_to(`c`)` **Else** `do_nothing()`.
    Else $\%_{r \neq \mathfrak{r}(x) \wedge r \neq \mathfrak{r}(y)}\%$
      If I Am Not $\mathfrak{r}(x)$ **And** I Am Not $\mathfrak{r}(y)$ **Then** `move_into_SEC()`
      Else `do_nothing()`.
  Else $\%_{|StartSet| = |revStartSet| = 1}\%$
    $Opposite = \{$Robot(s) Opposite To $\mathfrak{r}(x)$ And $\mathfrak{r}(y)$ $\}$;
    $Stationary = \{\mathfrak{r}(x), \mathfrak{r}(y)\} \cup Opposite$;
    If $r \in Stationary$ **Then**
      If I Am r **Then** `move_to(`c`)`. **Else** `do_nothing()`.
    Else $\%_{r \notin Stationary}\%$
      If $n - |Stationary| = 1$ **Then**
        If I Am Not in $Stationary$ **Then** `move_to(`c`)`
        Else `do_nothing()`.
      Else $\%_{n - |Stationary| \neq 1}\%$
        If I Am Not In $Stationary$ **Then** `move_into_SEC()`
        Else `do_nothing()`.
If More Than One Robot Is Inside SEC **Then**
  If I Am Inside SEC **Then** `move_to(`c`)` **Else** `do_nothing()`.

Note 2. As long as Subroutine 2 is executed, SA is general. In particular, all movements of the robots are straight towards c; thus, the string of angles does not change, and SEC remains invariant during the movements, until some robot reaches c.

In addition, note that Lines 13–14 as well as Lines 21–22 are only performed if the initial configuration runs into Lines 13–14 or Lines 21–22, respectively, since the algorithm does not create the corresponding configurations from any other initial configuration. In the following, we analyze all possible initial configurations that make the main algorithm call this subroutine.

(1) **All robots are on** SEC**, and** $|StartSet| = 2$. Then, $|revStartSet| = 0$ (by Lemma 5.5), and $\sphericalangle(\mathfrak{r}(x), c, \mathfrak{r}(y)) = 180°$ (by Lemma 5.6). In this case, the subroutine runs into Lines 3–5, where $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$ do not move, ensuring that the SEC remains invariant. All other robots are allowed to move inside SEC, but – due to function `move_into_SEC()` – no robot will reach c in one cycle; thus, SA remains general. Let $Allowed$ be the set of robots that are allowed to move inside SEC; since $n \geq 5$, $|Allowed| \geq 3$. Two cases can apply.

   a. In finite time, two or more robots from $Allowed$ move inside SEC. The subroutine runs Lines 30–31, where all robots inside SEC move towards c. By Note 2, these lines are executed until one ore more robots reach c (by Assumptions A1 and A2, this happens in a finite number of cycles). If only one robot reaches c first, Lines 8–9 of the main algorithm are performed, and Case (E) applies. Otherwise (more than one robot reaches c simultaneously), the robots reach a configuration where c is the (unique) point with strict multiplicity, and Case (SM) applies.

   b. In finite time, only one robot from $Allowed$, say r, moves inside SEC. Since, by construction, $r \neq \mathfrak{r}(x)$ and $r \neq \mathfrak{r}(y)$, then Lines 15–17 of the subroutine are executed. By `move_into_SEC()`, r does not reach c in one cycle. Furthermore, after the first movement inside SEC, it cannot move any more until at least another robot from $Allowed$ enters SEC. In finite time, this eventually happens, and previous case applies.

(2) **All robots are on** SEC**, and** $|revStartSet| = 2$. Then, $|StartSet| = 0$ (by Lemma 5.5), and, $\sphericalangle(\mathfrak{r}(x), c, \mathfrak{r}(y)) = 180°$ (by Lemma 5.6). The argument follows similarly to previous Case (1).

(3) **All robots are on** SEC**, and** $|StartSet| = |revStartSet| = 1$. The subroutine performs Lines 6–9. The robots in $Stationary$ do not move, and SEC remains invariant since c, by construction, is inside the convex hull of these robots (by Lemma 5.1). Furthermore, there are at most 2 opposite robots for $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$; thus, $|Stationary| \leq 4$. Since $n \geq 5$, at least one robot is allowed to move, and it moves into the inside of SEC. Two cases can apply.

   a. In finite time, two or more robots not in $Stationary$ move inside SEC. The subroutine runs Lines 30–31, and the same argument used in previous Case (1).a applies.

   b. In finite time, only one robot, say r, moves inside SEC. Since, by construction, $r \neq Stationary$, then Lines 23–29 of the subroutine are executed. By

`move_into_SEC()`, r does not reach c in one cycle. We distinguish the two possible cases.

i. If $n - |Stationary| = 1$, then note that, since $|Stationary| \leq 4$, this case can only occur if $n = 5$, and if there are two robots opposite to $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$. In this case, the subroutine executes Lines 24–26, where r, the only robot not in $Stationary$, moves towards c (r does not have to wait for any other robot to enter SEC, and all other robot do not move). Since SA stays general while r approaches c (by Note 2), Lines 24–26 are executed until, in a finite number of cycles, r reaches c. When this happens, Lines 8–9 of the main algorithm are performed, and Case (E) applies.

ii. If $n - |Stationary| \neq 1$, then note that, since $n \geq 5$ and $|Stationary| \leq 4$, then $n - |Stationary| \geq 2$. In this case, the subroutine executes Lines 27–29, where all robots not in $Stationary$ are allowed to move towards c. By `move_into_SEC()`, r (by construction already inside SEC) does not move until, in finite time, at least one other robot from $Stationary$ enters SEC. As soon as this happens, the subroutine runs Lines 30–31, and the same argument used in previous Case (1).a applies.

(4) **All robots except one, say** r**, are on** SEC**,** $|StartSet| = 2$ **(thus,** $|revStartSet| = 0$**), and** $r = \mathfrak{r}(x)$ **or** $r = \mathfrak{r}(y)$. The subroutine performs Lines 11–14: only r is allowed to move, and it moves towards c. By Note 1, this case applies until, in a finite number of cycles, r reaches c. Then, Lines 5–7 of the main algorithm are executed, and Case (C) applies.

(5) **All robots except one, say** r**, are on** SEC**,** $|revStartSet| = 2$ **(**$|StartSet| = 0$**), and** $r = \mathfrak{r}(x)$ **or** $r = \mathfrak{r}(y)$. The subroutine performs Lines 11–14. The argument follows as in previous Case (4).

(6) **All robots except one, say** r**, are on** SEC**,** $|StartSet| = 2$ **(**$|revStartSet| = 0$**),** $r \neq \mathfrak{r}(x)$**, and** $r \neq \mathfrak{r}(y)$. The subroutine performs Lines 15–17: an argument similar to the one used in previous Case (1).b applies.

(7) **All robots except one, say** r**, are on** SEC**,** $|revStartSet| = 2$ **(**$|StartSet| = 0$**),** $r \neq \mathfrak{r}(x)$**, and** $r \neq \mathfrak{r}(y)$. The subroutine performs Lines 15–17. The argument follows as in previous Case (6).

(8) **All robots except one, say** r**, are on** SEC**,** $|StartSet| = |revStartSet| = 1$**, and** $r \in Stationary$. The subroutine runs into Lines 21–22: r is allowed to move towards c, while all other robots do not move. By Note 1, this case applies until, in a finite number of cycles, r reaches c. Then, Lines 5–7 of the main algorithm are executed, and Case (C) applies.

(9) **All robots except one, say** r**, are on** SEC**,** $|StartSet| = |revStartSet| = 1$**,** $r \notin Stationary$**, and** $n - |Stationary| = 1$. The subroutine runs into Lines 24–26, and a similar argument to the one used in previous Case (3).b.i applies.

(10) **All robots except one, say r, are on SEC, $|StartSet| = |revStartSet| = 1$, $r \notin Stationary$, and $n - |Stationary| \neq 1$.** The subroutine performs Lines 27–29, and a similar argument to the one used in previous Case (3).b.ii applies.

(11) **More than one robot is inside SEC.** The subroutine runs into Lines 30–31, and a similar argument to the one used in previous Case (1).a applies.

Therefore, we can state the following

Theorem 5.3. *If the initial configuration of the robots is such that Subroutine 2 is executed, then in a finite number of cycles the robots gather at a point.*

5.6 Subroutine ManyStartingIndices

In this section, we describe subroutine `ManyStartingIndices()` (SubRoutine 3), called in Line 17 of the main algorithm. This case is executed when at the beginning SA is general and $|StartSet \cup revStartSet| \geq 3$. Therefore, there are more than two starting positions of $LexMinString$ in SA and $revSA$; hence, at least one of the two sets $StartSet$ and $revStartSet$ has cardinality strictly greater than one: let us assume, without loss of generality, that

$$|StartSet| \geq 2. \tag{3}$$

Moreover, let $StartSet = \{x_1, \ldots, x_l\}$, with $l \geq 2$ and $x_i < x_{i+1}$, for all $1 \leq i \leq l-1$ (i.e., $StartSet$ is sorted), and $SA(x_1) = \alpha_1, \ldots, \alpha_n = LexMinString$ (refer to Figure 12). Then, by definition of $StartSet$, $SA(x_i) = SA(x_1) = LexMinString$, for all $1 \leq i \leq l$. In particular, we have that

$$SA(x_2) = \alpha_{k+1}, \ldots, \alpha_n, \alpha_1, \ldots, \alpha_k, \tag{4}$$

with $k \geq 1$ the smallest integer such that Equation (4) holds. Then, $SA(x_3) = \alpha_{2k+1}, \ldots, \alpha_n, \alpha_1, \ldots, \alpha_{2k}$, and, in general, $SA(x_i) = \alpha_{(i-1)k+1}, \ldots, \alpha_n, \alpha_1, \ldots, \alpha_{(i-1)k}$, for all $1 < i \leq l$. That is, positions $j \cdot k$ in $SA(x_1)$, with $0 \leq j < \frac{n}{k} = l$ are all starting positions of $LexMinString$ (in the clockwise direction, according to the local orientation), and $|StartSet| = \frac{n}{k}$. We say that $SA(x_1)$ is *periodic* with minimum period length k, and $\alpha_1, \ldots, \alpha_k$ is a *period* of $SA(x_1)$. Furthermore, recalling that the characters in SA are angles, we observe that

Observation 5.1. *$SA(x_1)$ can be divided into $\frac{n}{k}$ equal periods, and the angles in each period sum up to $\beta = \frac{360^\circ}{\frac{n}{k}}$.*

We say that two robots r and r' are *equivalent (modulo periodic shift)* if $\sphericalangle(r, c, r')$ is a multiple of β. From the definition of period of SA, the following observation holds (refer to Figure 12):

Observation 5.2. *Let all robots be on SEC, and r be a robot. Then, r has $\frac{n}{k} - 1$ equivalent robots. Moreover, r and the robots equivalent to r form a regular $\frac{n}{k}$-gon, and c is inside this $\frac{n}{k}$-gon.*

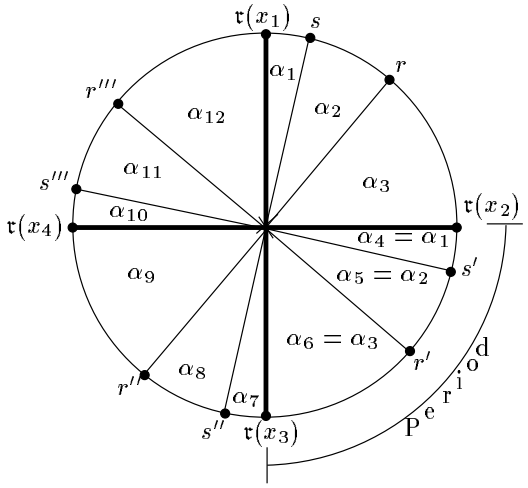


Figure 12: Example with $|StartSet|$=4, $SA(x_1) = SA(x_i) = LexMinString = \alpha_1, \ldots, \alpha_{12}$, and the period of $SA(x_1)$ is $\langle \alpha_1, \alpha_2, \alpha_3 \rangle$. There are $\frac{n}{k} = \frac{12}{3} = 4$ periods, and $\beta = \alpha_1 + \alpha_2 + \alpha_3 = 360°/\frac{n}{k} = 360°/\frac{12}{3} = 90°$. The thick lines represent the starting points of each of the four periods. Robots r, r', r'', and r''' are *equivalent*, as well as $\mathfrak{r}(x_1)$, $\mathfrak{r}(x_2)$, $\mathfrak{r}(x_3)$, $\mathfrak{r}(x_4)$, and s, s', s'' and s'''.

Obviously, $revSA$ is periodic with minimum period length k as well. Moreover, if $revStartSet \neq \emptyset$, we have $|StartSet| = |revStartSet| = \frac{n}{k}$. Based on the above definitions, we state the following

Lemma 5.7. *Assume that $StartSet \neq \emptyset$, that all robots are on SEC, and that the minimum period length of SA is $k \geq 3$. Then SEC remains invariant if all robots $\mathfrak{r}(x)$, with $x \in \{StartSet \cup revStartSet\}$, move inside SEC.*

Proof. It follows from the definition of the minimum period length, that all robots $\mathfrak{r}(x)$ such that $x \in StartSet$ are equivalent; similarly, all robots $\mathfrak{r}(x)$ such that $x \in revStartSet$ are equivalent.

Since $k \geq 3$, and $LexMinString$ starts in SA every k positions, there is at least one position x' in SA that is not $StartSet$; let r the robot associated to x'. It follows that neither r nor any of its equivalent robots are associated to positions that are in $StartSet \cup revStartSet$. Moreover, since by hypothesis all robots are on SEC, by Observation 5.2, r and its equivalent robots form a regular $\frac{n}{k}$-gon, and c is inside this $\frac{n}{k}$-gon. Thus, by Lemma 5.1, SEC does not change if all robots associated to positions in $StartSet \cup revStartSet$ move inside SEC. □

Next, we show that the configuration of the robots is totally symmetric if the period lenght is less than three.

Lemma 5.8. *If all robots are on SEC, $|StartSet \cup revStartSet| \geq 3$, and the minimum period length k is strictly smaller than 3, then the robots are in a totally symmetric configuration.*

Proof. Let $x_1 \in StartSet$ and $SA(x_1) = \alpha_1, \ldots, \alpha_n$. If $k = 2$, then $\alpha_1 = \alpha_3 = \ldots = \alpha_{2i-1} = \alpha$, and $\alpha_2 = \alpha_4 = \ldots = \alpha_{2i} = \beta$, for all $1 \leq i \leq n/2$ and n even. That is, $SA(x_1) = \alpha, \beta, \alpha, \beta, \ldots$; hence, it is totally symmetric.

Similarly, if $k = 1$, then $\alpha_1 = \alpha_2 = \ldots = \alpha_n$, and the lemma follows. □

In the following we show that gathering is achieved when at the beginning the configuration of the robots is such that this subroutine is called (i.e., in cases (A), (D), and (F) of Section 5.3). First, observe that

Note 3. As long as Subroutine 3 is executed, SA is general. In particular, all movements of the robots are straight towards c; thus, the string of angles does not change and SEC remains invariant during the movements, until some robot reaches c.

SubRoutine 3 ManyStartingIndices

```
    k :=Minimum Period Length of SA.
    If All Robots Are On SEC Then
        If I Am Associated To A Position In {StartSet ∪ revStartSet} Then
            move_into_SEC().
 5:     Else do_nothing().
    If Only One Robot r Is Inside SEC Then
        If k ≤ 2 Then
            If I Am r Then move_to(c) Else do_nothing().
        Else %k ≥ 3%
10:         If r = r(x)|r ∈ {StartSet ∪ revStartSet} Then
                If I Am Assoc. To A Pos. In StartSet ∪ revStartSet Then
                    move_into_SEC().
                Else do_nothing().
            Else %r is not associated to any of the positions in StartSet ∪ revStartSet%
15:             If I Am r Then move_to(c) Else do_nothing().
    If More Than One Robot Is Inside SEC Then
        If I Am Inside SEC Then move_to(c) Else do_nothing().
```

Note that Lines 7–8 as well as Lines 14–15 are only performed if the initial configuration runs into Lines 7–8 or Lines 14–15, respectively, since the algorithm does not create the corresponding configurations from any other initial configuration.

In the following, like in the case of one and two starting indices of LexMinString, we consider all possible initial configurations that make the main algorithm call this subroutine. Without loss of generality, we assume that $StartSet \neq \emptyset$.

(1) **All robots are on** SEC. The subroutine performs Lines 2–4, where all robots associated to a position in $StartSet \cup revStartSet$ are allowed to move inside SEC, while all other robots remain on SEC: let $Allowed$ be the set of robots allowed to move at this time (note that, by Equation (3), $|Allowed| \geq 2$). By Lemma 5.8, $k \geq 3$; therefore, by Lemma 5.7, SEC remains invariant if we move robots in $Allowed$ inside SEC. By definition of `move_into_SEC()`, none of the

robots allowed to move will reach c (in one cycle). Moreover, in finite time, either one or more than one robot enter inside *SEC*. We distinguish the two cases.

a. In finite time, two or more robots from *Allowed* move inside *SEC*. The subroutine runs Lines 16–17, where all robots inside *SEC* move towards c. By Note 3, this case applies until one ore more robots reach c (by Assumptions A1 and A2, this happens in a finite number of cycles). If only one robot reaches c first, Lines 8–9 of the main algorithm are performed, and Case (E) applies. Otherwise (more than one robot reaches c simultaneously), there is a (unique) point with strict multiplicity at c, and Case (SM) of the main algorithm applies.

b. If only one robot in *Allowed* leaves the circumference of *SEC*, say r, then the subroutine runs into Lines 10–13 (by Note 3, r is still associated to a position in $StartSet \cup revStartSet$). By definition of `move_into_SEC()`, r does not move until at least another robot associated to a position in $StartSet \cup revStartSet$ enters *SEC*. By Assumptions A1 and A2, this eventually happens in finite time, and, by Lemma 5.7, *SEC* does not change. According to the subroutine, at this time Lines 16–17 are executed, and the argument follows as in previous case.

(2) **All robots except one, say r, are on *SEC*, and $k \leq 2$.** The subroutine runs into Lines 7–8, where robot r moves towards c. By Note 3, this case applies until, in a finite number of cycles, r reaches c. When this happens, Lines 5–7 of the main algorithm are executed, and case (C) applies.

(3) **All robots except one, say r, are on *SEC*, $k \geq 3$, and r is associated to a position in $StartSet \cup revStartSet$.** The subroutine executes Lines 10–13: all robots associated in $StartSet \cup revStartSet$ are allowed to move inside *SEC* (by definition of `move_into_SEC()`, r does not move). The argument follows as in previous Case (1).b.

(4) **All robots except one, say r, are on *SEC*, $k \geq 3$, and r is not associated to any of the positions in $StartSet \cup revStartSet$.** The subroutine executes Lines 14–15: r moves straight towards c. By Note 3, this case applies until, in a finite number of cycles, r reaches c. At this point, Lines 5–7 of the main algorithm are executed, and Case (C) applies.

(5) **More than one robot is inside *SEC*.** The subroutine runs into Lines 16–17, and a similar argument to the one used in previous Case (1).a applies.

Therefore, we can state the following

Theorem 5.4. *If the initial configuration of the robots is such that Subroutine 3 is executed, then in a finite number of cycles the robots gather at a point.*

By Theorems 5.1–5.4, we can finally state the following

Result 5. Five or more robots that can detect multiplicity and that are not in a totally symmetric configuration at the beginning can always gather at a point in a finite number of cycles.

6 Conclusion

In this paper, we studied the GATHERING PROBLEM for a set of autonomous mobile robots. This problem is unsolvable for robots that cannot detect multiplicities. For $n = 2$ the problem is still unsolvable, unless they can detect collisions. For $n = 3$ we observed that the robots can choose as gathering point the Weber point of their initial positions. For $n = 4$ we presented an ad-hoc algorithm, that, in the general case, gather the robots in the intersection of the two diagonals of the quadrilater having as vertices the position of the robots at the beginning.

For more than four robots, we briefly outlined the solution for the GATHERING PROBLEM when the robots are at the beginning in a biangular configuration; this algorithm covers also the case when the robots are in a totally symmetric configuration at the beginning. In the more general case when the robots are not in a totally symmetric configuration at the beginning, we presented Algorithm 4, that solves the problem in finite time. This algorithm gathers the robots either at the center of the smallest circle enclosing the starting positions of the robots, or at a point on the circumference of the *SEC*.

Hence, we presented two different algorithms, which together cover all possible initial configurations. However, these two algorithms cannot be simply combined to solve the entire GATHERING PROBLEM: let the robots' initial configuration be non–biangular. Hence, they start executing Algorithm 4. During the run of the algorithm, it may happen that the robots form a biangular configuration. Since the robots act completely asynchronously, some of them may observe this biangular configuration, while others do not. Thus, some of the robots move to the center of biangularity, while others still perform Algorithm 4. Hence, they might not gather at a point. Therefore, the remaining challenge is to design an algorithm that solves the GATHERING PROBLEM with $n \geq 5$ robots for *any* initial configuration.

In Theorem 2.1, we stated that the GATHERING PROBLEM is solvable only if the robots can detect multiplicities. Obviously, instead of multiplicity detection, other abilities may be considered in future work. For instance, it would be interesting to explore the relationship between memory and solvability of the assigned tasks, or to study the presence of some kind of communication among the robots.

Acknowledgements

We would like to thank Paola Flocchini and Nicola Santoro for the intersting discussions, and Stephan Eidenbenz, Vincenzo Gervasi, Zsuzsanna Lipták, Konrad Schlude, and Peter Widmayer for the helpful comments.

References

[1] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. A Distributed Memoryless Point Convergence Algorithm for Mobile Robots with Limited Visibility. *IEEE Transaction on Robotics and Automation*, 15(5):818–828, 1999.

[2] T. Balch and R. C. Arkin. Behavior-based Formation Control for Multi-robot Teams. *IEEE Transaction on Robotics and Automation*, 14(6), December 1998.

[3] M. Cieliebak and G. Prencipe. Gathering Autonomous Mobile Robots. In *Proceedings of the* 9^{th} *International Colloquium On Structural Information And Communication Complexity (SIROCCO 9)*, pages 57–72, June 2002.

[4] E. J. Cockayne and Z. A. Melzak. Euclidean Constructibility in Graph-minimization Problems. *Mathematical Magazine*, 42:206–208, 1969.

[5] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots. In *Proceedings of the* 10^{th} *Annual International Symposium on Algorithms and Computation (ISAAC '99)*, volume 1741 of *LNCS*, pages 93–102, 1999.

[6] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Autonomous Mobile Robots With Limited Visibility. In *Proceedings of the* 18^{th} *International Symposium on Theoretical Aspects of Computer Science (STACS 2001)*, volume LNCS 2010, pages 247–258, 2001.

[7] D. Jung, G. Cheng, and A. Zelinsky. Experiments in Realising Cooperation between Autonomous Mobile Robots. In *ISER*, 1997.

[8] M. J. Matarić. Designing Emergent Behaviors: From Local Interactions to Collective Intelligence. In *From Animals to Animats 2: Int. Conf. on Simulation of Adaptive Behavior*, pages 423–441, 1993.

[9] Z. A. Melzak. On the Problem of Steiner. *Canadian Mathematical Bulletin*, 4(2):143–148, May 1961.

[10] G. Prencipe. CORDA: Distributed Coordination of a Set of Autonomous Mobile Robots. In *ERSADS 2001*, pages 185–190, 2001.

[11] G. Prencipe. *Distributed Coordination of a Set of Autonomous Mobile Robots.* PhD thesis, Università di Pisa, 2002. http://sbrinz.di.unipi.it/~peppe/tesi.ps.

[12] G. Prencipe. On the Effect of Synchronicity on the Behavior of Autonomous Mobile Robots. *Submitted to Theory of Computing Systems*, 2002.

[13] I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *Siam Journal of Computing*, 28(4):1347–1363, 1999.

[14] E. Weiszfeld. Sur le Point Pour Lequel la Somme Des Distances de n Points Donnés Est Minimum. *Tohoku Mathematical*, 43:355–386, 1936.

[15] E. Welzl. Smallest Enclosing Disks (Balls and Ellipsoids). *LNCS*, 555:359–370, 1991.