Solving the Robots Gathering Problem

Mark Cieliebak[1], Paola Flocchini[2], Giuseppe Prencipe[3], and Nicola Santoro[4]

[1] ETH Zurich, cieliebak@inf.ethz.ch
[2] University of Ottawa, flocchini@site.uottawa.ca
[3] University of Pisa, prencipe@di.unipi.it
[4] Carleton University, santoro@scs.carleton.ca

Abstract. Consider a set of $n > 2$ simple autonomous mobile robots (decentralized, asynchronous, no common coordinate system, no identities, no central coordination, no direct communication, no memory of the past, deterministic) moving freely in the plane and able to sense the positions of the other robots. We study the primitive task of gathering them at a point not fixed in advance (Gathering Problem). In the literature, most contributions are simulation-validated heuristics. The existing algorithmic contributions for such robots are limited to solutions for $n \leq 4$ or for restricted sets of initial configurations of the robots. In this paper, we present the first algorithm that solves the Gathering Problem for *any* initial configuration of the robots.

1 Introduction

We consider a distributed system of autonomous mobile robots that are able to freely move in the two-dimensional plane. Due to their autonomy, the coordination mechanisms used by the robots to perform a task (i.e., solve a problem) must be totally *decentralized*, i.e., no central control is used. The problem we consider is *gathering* (or rendez-vous, or point-formation): all robots must gather at one point; the choice of the point is not fixed in advance.

Gathering is one of the basic interaction primitives in systems of autonomous mobile robots, and has been studied in robotics and in artificial intelligence [4,9, 11]. 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 coordination problems have motivated *algorithmic* investigations, and the problems have also been approached from a *computational* point of view [2,7,8,12, 14].

The solution to the Gathering Problem obviously depends on the capabilities of the robots. The research interest is on a very weak model of autonomous robots: the robots are anonymous (i.e., identical), have no common coordinate system, are oblivious (i.e., they do not remember previous observations and calculations), and have no means of direct communication. Initially, they are in

J.C.M. Baeten et al. (Eds.): ICALP 2003, LNCS 2719, pp. 1181–1196, 2003.

a waiting state. They wake up independently and 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. Details of the model are given in Section 2. For these robots, the GATHERING PROBLEM is defined as follows:

Definition 1. *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.*

This GATHERING PROBLEM is *unsolvable* for such weak robots [13]; this is rather surprising considering the fact that a variety of other tasks (e.g. forming a circle) are solvable. Also, if the robots are asked only to move "very close" to each other, this task is easily solved: each robot computes the center of gravity[2] of all robots, and moves towards it.

The reason the same solution (i.e., moving towards the center of gravity) does not work for the GATHERING PROBLEM is because the center of gravity is not invariant with respect to robots' movements towards it. Recall that the robots act independently and asynchronously from each other, and that they have no memory of the past; once a robot makes a move towards the center of gravity, the position of the center of gravity changes; hence a robot (even the same one) observing the new configuration will compute and move towards a different point.

An obvious solution strategy would then be to choose as destination a point that, unlike the center of gravity, is *invariant* with respect to the robots' movements towards it. The only known point with such a property is the *Weber* (or *Fermat* or *Torricelli*) *point*: the unique point in the plane that minimizes the sum of the distances between itself and all positions of the robots [10,15]. This point does not change when moving any of the robots straight towards it. Unfortunately, it has been proven in [3] that the Weber point is not expressible as an algebraic expression involving radicals since its computation requires finding zeroes of high-order polynomials even for the case $n = 5$ (see also [6]). In other words, the Weber point is *not computable* by radicals for $n \geq 5$ [3], and thus it cannot be used to solve the GATHERING PROBLEM.

The problem becomes solvable if we change the nature of the robots: if we assume a common coordinate system, gathering is possible even with limited visibility [8]; if the robots are synchronous and movements are instantaneous, gathering has a simple solution [14] and can be achieved even with limited visibility [2]. On the other hand, without changing the robots' nature, they clearly must have some additional ability to solve the GATHERING PROBLEM. One such ability is *multiplicity detection*: a robot can detect whether at a point there is none, one, or more than one robot; if there is more than one robot, we say that

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

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

there is *strict multiplicity* at that point. In the following, we will assume that the robots can detect multiplicities.

Even with multiplicity detection, the problem is surprisingly difficult and was, up to now, unsolved. It is actually unsolvable for $n = 2$ robots [13,14]. Simple solution algorithms exist for $n = 3$ and $n = 4$ robots. For $n \geq 5$ there are two *partial* solutions [5], i.e., algorithms that work for restricted sets of initial configurations. In particular, the first one works if the robots are initially in a biangular configuration (i.e., there exists a point c, and ordering of the robots, and two angles α, β such that the angles between adjacent robots w.r.t. c are either α or β, and the angles alternate; refer to Section 2 and Figure 2); the second algorithm works if in the initial scenario the positions of the robots do not form a regular n-gon (i.e., all robots are on a circle and the distances between each two adjacent robots are equal). Although the two sets of configurations together cover all possible input configurations, the two algorithms can not be integrated nor combined to solve the GATHERING PROBLEM in general.

In this paper, we present the first algorithm that solves the GATHERING PROBLEM for *any* initial configuration of the robots; all calculations performed by the robots can be computed by radicals. Due to space limitations, we only sketch the algorithm and the main ideas for its correctness. The complete algorithm and detailed proofs can be found in the full version of this paper.

2 Terminology, Notation, and Basic Tools

In this section, we introduce terminology and notation, and define the basic concepts used in our algorithm.

Autonomous Mobile Robots

A robot is a mobile computational unit provided with sensors, and it is viewed as a point in the plane. Once activated, the sensors return the set of all points in the plane occupied by at least one robot. In particular, for each such point the sensor outputs whether one or more than one robot is located there (*multiplicity detection*). This forms the current *local view* of the robot. The local view of each robot also 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). There is no a priori agreement among the robots on the unit of length, the origin, or the 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 (the same for all robots), based only on its local view of the world. It then *moves* towards the destination point (*Move*); if the destination point is the current location, the robot stays still. A move may stop before

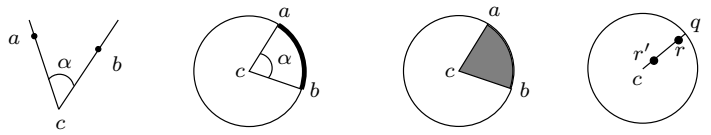

Fig. 1. (a) Convex angle $\alpha = \sphericalangle(a, c, b)$. (b) Arc (thick line) and (c) sector (grey part) defined by $\sphericalangle(a, c, b)$. (d) Two robots, r and r', on the same radius.

the robot reaches its destination, e.g. because of limits to the robot's motion energy. The robot then returns to the waiting state. 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 state 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 observation nor computations performed in previous cycles. 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.

Notation and Definitions

Basic Notation. 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 time t.

For the following definitions, refer also to Figure 1. 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 that 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 $\mathcal{C}$ is denoted by $arc(\alpha)$, 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 say that r *is on* $\mathcal{C}$ if $dist(rc) = Rad$, where $dist(ab)$ denotes the Euclidean distance between

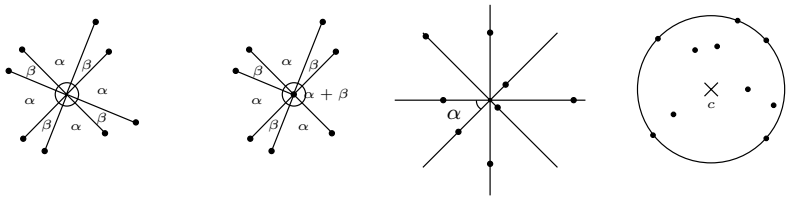

Fig. 2. (a) General biangular and (b) degenerated biangular configuration of 8 points. (c) General equiangular configuration. (d) The smallest enclosing circle of 10 points on the plane.

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 $r' \in [c, q]$.

Biangular Configurations. A set of n robots is in *general biangular configuration* if there exists a point b, 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. b, and the angles alternate (see Figure 2.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 (without r) 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 2.b). A general biangular configuration becomes degenerated if one of the robots, namely r, moves to the center b.

Similarly, we say that the robots are in a *general equiangular configuration* if there exists a point e, the *center*, an ordering of the robots, and an angle α such that each two adjacent robots form an angle α w.r.t. e (see Figure 2.c). Note that equiangular configurations can be "almost" considered a special case of biangular configurations: the only difference is that in a biangular configuration there is always an even number of robots, while in an equiangular configuration there can be an odd number of robots. Hence, from now on we will only refer to biangular configurations.

If a set of $n \geq 3$ points P is in general or degenerated biangular configuration, then the center of biangularity b is unique, can be computed in polynomial time, and is invariant under straight movement of any of the points in its direction; that is, it does not change if any of the points move towards b [1].

Smallest Enclosing Circles. Given a set of n distinct points P in the plane, the *smallest enclosing circle* of the points is the circle with minimum radius such that all points from P are inside or on the circle (see Figure 2.d). 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 polynomial time [16].

Obviously, the smallest enclosing circle of P remains invariant if we remove all or some of the points from P that are inside $SEC(P)$. In fact, the following lemma shows that we can even remove all but at most three points from P without changing $SEC(P)$.

Lemma 1. *Given a set P of n points, there exists a subset $S \subseteq P$ such that $|S| \leq 3$ and $SEC(S) = SEC(P)$.*

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 3.a):

`Compute_SA(`p_k`)`

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

Here, all angles are oriented clockwise (note that the robots do not have a common coordinate system; however, each robot can locally distinguish between a clockwise and counterclockwise orientation). The successor of p, computed by `Succ(`p`)`, is (refer to Figure 3.b)

- 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
- 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 unique successors, and thus `Compute_SA(`p_k`)` defines 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, and we write $p = \mathfrak{r}(\alpha)$. Alternatively, since α is stored in SA, say at position i (i.e., $SA[i] = \alpha$), we denote the point associated to α by $\mathfrak{r}(i)$, saying that $\mathfrak{r}(i)$ is the point associated to position i in SA. We define the *reverse string of angles* $revSA$ in an analogous way: it is the string of angles with all angles counterclockwise oriented (i.e., $revSA$ is the reverse of SA). We say that SA (resp. $revSA$) 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*.

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),

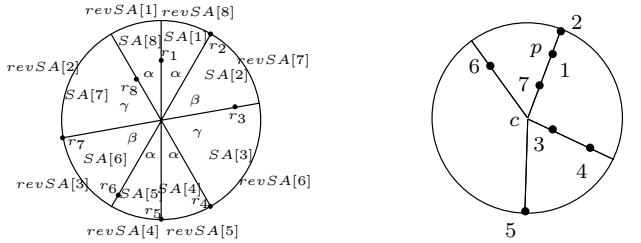

Fig. 3. (a) String of angles computed by `Compute_SA(`r_1`)`. With $\alpha = 25°$, $\beta = 60°$ and $\gamma = 70°$, we have $SA(r_1) = \langle \alpha, \beta, \gamma, \alpha, \alpha, \beta, \gamma, \alpha \rangle = \langle 25°, 60°, 70°, 25°, 25°, 60°, 70°, 25° \rangle$, $LexMinString = \langle \alpha, \alpha, \beta, \gamma, \alpha, \alpha, \beta, \gamma \rangle$, $\mathfrak{r}(SA[3]) = \mathfrak{r}(\gamma) = \mathfrak{r}(3) = r_3$, $StartSet = \{4, 8\}$, and $revStartSet = \emptyset$. (b) Routine `Succ(`p`)` with clockwise orientation. The points are numbered according to routine `Succ()`; that is `Succ(1)`=2, `Succ(2)`=3, and so on. Note that `Succ(7)`=1

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.

Robot Motion and Critical Points

In our algorithm, we use four different types of "move" operations; in each, when a robot moves, it moves in a straight line.

The basic operation is `moveTo(`p`)`, where a robot r moves towards point p (recall that, although restricted by assumption **A2**, the robot may enter the waiting state before reaching p).

In the operation `moveToIfFreeWay(`p`)`, the robot r moves towards p only if no other robot is between r and p; otherwise, r does not move at all. This operation is used to avoid that the moving robot creates an (unintended) point with strict multiplicity. Note that, if all robots in the system are moving towards p and only this type of moves are executed, then strict multiplicity can only occur at p.

The remaining two types of movement are crucial to control the swap of a non-biangular configuration into a biangular one, due to robots' movements. To introduce them, we need the notion of *critical points*, defined as follows:

Definition 2. *Given n robots and a point p in the plane, a point x is a* critical point *for the movement of robot r towards p if $x = p$, or if x is on the half-line from p to r and the configuration of the robots becomes biangular when r is at position x.*
A pair of points (y, z) is a critical pair *for the movements of robots r' and r'' towards destinations p' and p'' respectively, if $(y, z) = (p', p'')$, or if y is on the*

half-line from p' to r', z is on the half-line from p'' to r'', and the configuration of the robots is biangular when r' is at position y and r'' is at position z.

The operation `moveStepwiseTo(`p`)` requires the robot r to first compute all critical points for its movement towards p, and then to move towards the first critical point on its way towards and stop there.

With operation `moveStepwiseTo(`$(r', p'), (r'', p'')$`)` we coordinate the movement of two robots r' and r'' which move in the direction of points p' and p'', respectively. We compare the number of critical points between the robots and their destinations. The robot with most critical points ahead is allowed to move; if they have the same number, they both move. Once allowed, if the robot is between two critical points it moves to the next one; if it is already at a critical point it moves towards half the distance to the next critical point.

Finally, given a circle C with center c, we extend our four types of move operations and allow robots to move onto or away from C. In particular, we say that a robot moves to circle C (`moveTo(`C`)`, `moveToIfFreeWay(`C`)`, `moveStepwiseTo(`C`)`) if the destination point of the robot is the intersection of C and the half-line starting from the center of C and going through the position of the robot (note that the robot does not move at all if it is already at this intersection point). Moreover, we define what a movement into the inside of circle C is (`moveTo(into` C`)`, `moveToIfFreeWay(into` C`)`, `moveStepwiseTo(into` C`)`): if the robot is already inside C, it does not move at all. Otherwise, it moves to the point p that is half on its way towards c, the center of the circle. For two robots r' and r'', we define `moveStepwiseTo(`r', r'', C`)` and `moveStepwiseTo(`$r', r'',$ `into` C`)` accordingly.

3 The Solution Algorithm

In this section, we describe the algorithm that solves the Gathering Problem for arbitrary initial configurations of $n \geq 5$ robots, and discuss its correctness.

3.1 Description

At a high level, the strategy of the algorithm is as follows. Initially all robots are in distinct locations; that is, in the initial configuration, there is no point with strict multiplicity. Our algorithm ensures that at any time during the execution there is at most one point with strict multiplicity; moreover, such a point will eventually be generated. Once this occurs, the robots that are already at that point remain there, while all other robots move towards this unique point.

If the (initial) configuration is biangular, then all robots move towards the center of biangularity. The future configuration remains biangular until two (or more) robots reach the center. When this occurs, a unique point with strict multiplicity has been created.

In all other configurations, we select a strict subset of the robots; the selection is done using the string of angles of the robots w.r.t. the center of their smallest

enclosing circle. If we can elect a unique robot, it will go to some other robot creating a unique point with strict multiplicity. Otherwise, the selected robots move towards the center of the smallest enclosing circle, ensuring that the circle does not change because of these movements. If no biangular configuration is created during these movements, two (or more) robots reach the center of the circle, and we have a unique point with strict multiplicity.

One of the difficult and crucial components of the algorithm is the use of appropriate move operations to ensure the following: if a biangular configuration is created during the movements of some robots, then *all* robots have to become aware of it in their next *Look* state, ensuring that they will gather at the center of biangularity. The difficulty arises from the asynchrony, obliviousness and autonomy of the robots; the component is crucial to avoid that some robots move towards the center of biangularity while others still move towards the center of the circle (possibly destroying biangularity).

The main algorithm is shown in Algorithm 1. In the algorithm we use four different subroutines; their behavior differs depending on the value of s, the cardinality of the set $StartSet \cup revStartSet$ (therefore, s denotes the number of starting positions of $LexMinString$ in SA and $revSA$).

Algorithm 1 Algorithm GATHER

$\mathcal{Z}$:= Observed Configuration;
SEC := Smallest Enclosing Circle of all robots;
c := Center of SEC;
$InnerC$:= Circle with center c and radius $\frac{\text{radius of } SEC}{2}$;
Case $\mathcal{Z}$ **Is Such That:**
- *There is One point m with strict multiplicity:*
  `moveToIfFreeWay(`m`).`
- *The robots are in general (resp. degenerated) biangular configuration:*
  b := Center of general (resp. degenerated) biangularity;
  `moveToIfFreeWay(`b`).`
- *default:*
  If No robot is at c **Then**
    SA := (`Compute_SA`); %String of angles of all robots%
    $StartSet, revStartSet$:= Indices Where Lex. Minimal String Starts;
    $s := |StartSet \cup revStartSet|$;
    If SA is general **Then** `Routine1`. **Else** `Routine2`.
  Else %One robot r is at c%
    r := robot at c;
    SA^- := String of angles of all robots except r;
    $StartSet^-, revStartSet^-$:= Indices Where Lex. Minimal String Starts;
    If SA^- is general **Then** `Routine3`. **Else** `Routine4`.

In the following, we first discuss the main properties of $LexMinString$, and then we sketch the correctness proof of the algorithm.

3.2 Properties of *LexMinString*

1. One Starting Position of LexMinString ($s = 1$): Let $StartSet \cup revStartSet = \{x\}$ and $SA(x) = \alpha_1, \ldots, \alpha_n$; then $revSA(x) = \alpha_n, \ldots, \alpha_1$, and the following holds:

Lemma 2. *If $StartSet \cup revStartSet = \{x\}$, then either $SA(x) = LexMinString$ or $revSA(x) = LexMinString$.*

This implies that there is a unique starting position and a unique direction for $LexMinString$, yielding a unique ordering of the robots. If all robots are on SEC, then we can use this ordering and Lemma 1 to define operation `ElectOne()` to elect the first robot r such that SEC remains invariant if r is moved to the inside of SEC. If more than one robot is inside SEC, then `ElectOneInside()` is used to elect a unique robot that is already inside SEC (again, using the uniqueness of $LexMinString$).

2. Two Starting Positions of LexMinString ($s = 2$): Let $StartSet \cup revStartSet = \{x, y\}$. The following lemma shows that $LexMinString$ can start in each position in only one direction.

Lemma 3. *If $StartSet \cup revStartSet = \{x, y\}$, then it is not possible that $SA(x) = revSA(x) = LexMinString$ or $SA(y) = revSA(y) = LexMinString$.*

If $LexMinString$ starts in x and y in the same direction, then the angle between these two positions w.r.t. c is 180°. Moreover, for every robot there is a partner such that their angle is 180°. Recall that $\mathfrak{r}(x)$ is the robot associated with index x. Using the starting positions and the direction of $LexMinString$, we define `ElectTwo()` as follows: if $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$ are on SEC, then we elect the "next" pair of robots with an angle of 180°; otherwise we elect $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$ themselves.

If $LexMinString$ starts in x and y in opposite direction, say $x \in StartSet$ and $y \in revStartSet$, then let γ be the angle between $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$ w.r.t. c. If $\gamma = 180°$, then `ElectTwo()` elects the first two robots, according to the starting positions and directions of $LexMinString$, that are not both on SEC. If $\gamma < 180°$, we define the *opposite robots* of $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$ to be one or two robots in the half of SEC where $\mathfrak{r}(x)$ and $\mathfrak{r}(y)$ are not (see Figure 4): let ℓ be the line that bisects γ. Then ℓ is a symmetry line for the angles of the robots w.r.t. c. We choose either the robot r that is on line ℓ, if such a robot exists, or we choose the two robots u and v that are closest to ℓ (in terms of their angles w.r.t. c). 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). Thus, `ElectTwo()` can elect two appropriate robots such that SEC remains invariant if they move (using Lemma 1).

Finally, we define routine `ElectPairInside()` that elects the "first" pair of robots that is inside SEC. Again, the ordering of the robots is given by the starting positions and orientations of $LexMinString$.

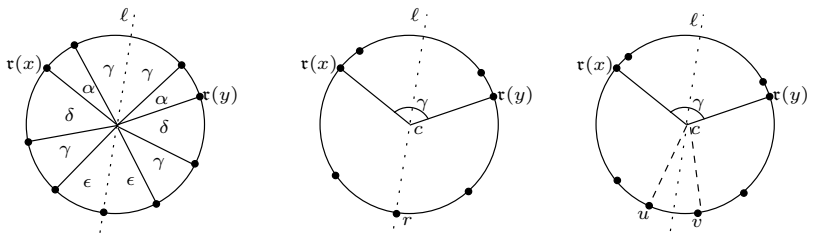

Fig. 4. (a) The line ℓ that runs through c and bisects $\gamma = \sphericalangle(\mathfrak{r}(x), c, \mathfrak{r}(y)) = 2\epsilon + 2\gamma$ is a symmetry axis for the angles that the robots form w.r.t. c. In the depicted example, $x \in StartSet$, $y \in revStartSet$, and $SA(x) = revSA(y) = LexMinString = \langle \alpha, \gamma, \gamma, \alpha, \delta, \gamma, \epsilon, \epsilon, \gamma, \delta \rangle$. (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 $\mathfrak{r}(y)$.

3. Many Starting Positions of LexMinString ($s > 2$): Let $StartSet = \{x_1, \ldots, x_l\}$. Then SA and $revSA$ are periodic. Moreover, if k is the minimum length of a period of SA, then we can divide SA into $\frac{n}{k}$ equal periods, and the angles in each period sum up to $\gamma = \frac{360^\circ}{\frac{n}{k}}$. If the period length is one or two, then the configuration is biangular; hence we can exclude this case in the following, since it is covered in Lines 8–10 of the main algorithm.

We say that two robots r and r' are *equivalent (modulo periodic shift)* if they have the same position in different periods, i.e., if $\sphericalangle(r, c, r')$ is a multiple of γ (see the example depicted in Figure 5). If all robots are on SEC, then for any robot r, there are $\frac{n}{k} - 1$ equivalent robots, and they form a regular $\frac{n}{k}$-gon with c inside. Thus, if at least one robot and all its equivalent robots remain on SEC, then SEC remains invariant (using Lemma 1).

Lemma 4. *If $StartSet \neq \emptyset$, all robots are on SEC, and the minimum period length of SA is $k \geq 3$, then SEC remains invariant when all robots $\mathfrak{r}(x)$, with $x \in \{StartSet \cup revStartSet\}$, move inside SEC.*

Observe that if all robots are on SEC, then equivalent robots cannot be distinguished, hence they act in the same way. In the case of one or two starting positions of $LexMinString$, we were able to elect one or two robots to move, and we used stepwise movements to ensure that these robots stop when the configuration becomes biangular. If there are many starting positions of $LexMinString$, we do not need to apply stepwise movements, as shown by the following lemma.

Lemma 5. *Given a non biangular configuration of the robots such that SA is periodic, then moving any subset of the robots towards c cannot make the configuration become biangular.*

To see this, observe that c is the Weber point of the robots, and that the center of biangularity, if it exists, is the Weber point as well. Thus, since the

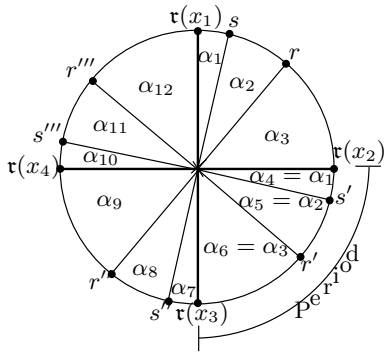

Fig. 5. Example with $|StartSet|$=4, $SA(x_1) = SA(x_i) = LexMinString = \alpha_1, \ldots, \alpha_{12}$, and the period of $SA(x_1)$ is $\langle \alpha_1, \alpha_2, \alpha_3 \rangle$. There are $\frac{n}{k} = \frac{12}{3}$ periods, and $\gamma = \alpha_1 + \alpha_2 + \alpha_3 = \frac{360^\circ}{4}$. Thick lines represent the starting points of each of the four periods. Robots r, r', r'' and r''' are *equivalent*, as well as $\mathfrak{r}(x_1)$, $\mathfrak{r}(x_2)$, $\mathfrak{r}(x_3)$ and $\mathfrak{r}(x_4)$, and s, s', s'' and s'''.

Weber point is unique, the robots cannot swap into a biangular configuration if there was none before. This lemma implies that the robots cannot create a biangular configuration while they move towards or away from c, hence we do not need to introduce a stepwise movement in this case.

Correctness Proof (Sketch)

The first thing a robot does when it starts its computation is to check whether there is a point p in the plane with strict multiplicity. If this is the case, the robot simply moves there. Point p will be the final gathering point (Lines 6–7).

Otherwise, the robots check whether the observed configuration of the robots is biangular. In this case, the center of biangularity b is computed, and the robots move there using `moveToIfFreeWay(`b`)`. As long as none of the robots reaches b, the configuration remains general biangular; hence the algorithm continues to move all robots towards b. By Assumptions A1 and A2, in a finite number of cycles, at least one robot reaches b. If only one robot reaches b, then the configuration becomes degenerated biangular. In this case, the center of degenerated biangularity[3] is again b, and all robots continue moving towards b. As soon as two robots reach b, there is a unique point with strict multiplicity, and all robots will gather there.

If the observed configuration is not biangular, then the SEC and its center c are computed. The algorithm distinguishes four cases.

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

(A) **There is no robot at c, and SA is general**. `Routine1` is called, which behaves differently depending on the value of s, the cardinality of $StartSet \cup revStartSet$.

If $s = 1$, then a unique robot r is elected, and it moves stepwise[4] towards c. Robot r is chosen such that SEC does not change during its movement. When the movement stops, either the configuration is biangular, and Line 8 of the main algorithm applies; or `Routine1` is called again (with r – the only robot inside SEC – again elected), until r reaches c, and `Routine3` applies.

If $s = 2$, at first all robots that are inside SEC move to the circumference of SEC (by repeatedly calling `ElectPairInside()`). Afterwards, only the two robots elected by `ElectTwo()` are allowed to move, and they move towards c. All movements are stepwise, and there are always at most two moving robots, either the robots run into a biangular configuration and stop (Line 8 of the main algorithm then applies), or one of them reaches c and `Routine3` is called, or the two elected robots reach c simultaneously, and c becomes the unique point with strict multiplicity. In the last two cases, c will be the final gathering point.

If $s > 2$, first all robots associated to indices in $StartSet \cup revStartSet$ are elected. Then, all robots that are not elected and that are inside SEC are moved towards the circumference of SEC. Afterwards, all elected robots (and only these) move towards[5] c (without changing SEC, by Lemma 4), with the only restriction that an elected robot can reach c only if all other elected robots are already inside SEC (note that two robots inside SEC would be sufficient). This is achieved by first calling routine `moveTo(into` C`)`. In a finite number of cycles at least two robots reach c, creating strict multiplicity there.

(B) **There is no robot at c, and SA is degenerated**. `Routine2` is called. Recall that, if SA is degenerated, then there is at least one radius of SEC with more than one robot on it. Therefore, due to our definition of SA, the lexicographically minimal string of angles always start with zeros. Moreover, on each radius with at least two robots, one robot is already inside SEC. Similarly to previous cases, different actions are taken depending on the value of s.

If $s = 1$, then the subroutine elects a unique radius rad that has at least two robots lying on it. Let $StartSet = \{x\}$ (the case $revStartSet = \{x\}$ is handled similarly), and rad_x be the radius where $\mathfrak{r}(x)$ lies (i.e., $[c, \mathfrak{r}(x)]$). Then rad can be chosen as the first radius with at least two robots on it, starting from rad_x and according to the ordering of the robots established by SA. Let r and r' be the first two innermost robots on rad. Then r moves stepwise towards r', while all other robots do not move. In a finite number of cycles, either a biangular configuration is reached (r stops at the first critical

[4] Recall that stepwise movement implies that r stops at its first critical point.

[5] Some of them can already be inside SEC, and others are still on the circumference of SEC.

point on its path towards r') and Line 8 of the main algorithm applies, or r reaches r' and a unique point with strict multiplicity is created.

If $s = 2$, the algorithm works similar to Case (A), except that all operations are done with respect to *InnerC* instead of *SEC*. In particular, first the robots that are inside *InnerC* move out of *InnerC*. If we would move these robots simply to the circumference of *InnerC*, we could obtain unintended points with strict multiplicity, since all robots on the same radius would end up at the same point on *InnerC*. Therefore, we define a sufficient number of distinct positions "just outside" *InnerC* (using the radius of *SEC*) where we move the robots that are on the same radius. Thereby, we ensure that the innermost robots will end up on *InnerC*. Afterwards, the two robots elected by `ElectTwo()` are allowed to move, and they move stepwise towards c, and in a finite number of cycles at least one of them reaches c.

If $s > 2$, then SA is periodic (see paragraph on $s > 2$). Again, the algorithm works similar to Case (A), except that all operations are done with respect to *InnerC* instead of *SEC*.

(C) **There is exactly one robot r at c, and SA^- (the string of angles of all robots except r) is general**. `Routine3` is called. If r is the only robot inside SEC, then r chooses an arbitrary robot q on *SEC* and moves stepwise towards it. By this movement, the string of angle becomes degenerated, since r and q are on the same radius. Hence, by (B), r continues to move towards q. If no critical points are on its path towards q, in a finite number of cycles r reaches q and a unique point with strict multiplicity is obtained. Otherwise, r stops at the first critical point it meets. Then a biangular configuration is obtained, and Line 8 of the main algorithm applies.

If there are only two robots r and r' inside *SEC* (with r at c), then r' moves stepwise towards c. The argument follows similarly to the previous paragraph.

If more than two robots are inside *SEC* and SA^- is periodic except for one gap[6], then all robots inside SEC move towards c. By Lemma 5, no biangularity can occur.

If more than two robots are inside *SEC* and SA^- is not periodic except for one gap, then the routine behaves similarly to `Routine1`. The only difference is that in this case all the operations are done using SA^- instead of SA; that is, robot r is ignored.

(D) **There is exactly one robot r at c, and SA^- (the string of angles of all robots except r) is degenerated**. `Routine4` is called. If r is not the only robot inside *InnerC*, then this routine is similar to Routine 3, except that all operations refer to *InnerC* instead of *SEC*. Otherwise, if r is the only robot inside *InnerC*, then it chooses some an arbitrary index q in $StartSet^- \cup revStartSet^-$. Note that q is always associated to a position in SA where $LexMinString$ starts. Robot r moves stepwise towards $\mathfrak{r}(q)$, while all other robots do not move. As soon as r leaves c, a unique starting

[6] That is, the string would be periodic if r — the robot at c — would not be at c, but somewhere inside SEC.

position of $LexMinString$ is obtained in the positions associated to r, since an angle with 0° has been added. Thus, SA is degenerated with no robot at c, and r will be chosen again to move on to q due to Case (B) above.

4 Conclusion

We have presented a deterministic algorithm for the GATHERING PROBLEM for $n \geq 5$ robots that works for all initial configurations of the robots. Some interesting questions are still open. For example, it is not known which other abilities, other than multiplicity detection, would allow the weak robots to solve the GATHERING PROBLEM.

It is known that changing the nature of the robots (e.g. by synchronizing them, or by adding common knowledge on the coordinate system) enables solvability. It is still not known if (and how) removing obliviousness would have the same effect. It would be interesting to explore the relationship between memory and solvability or, for that matter, to study the impact of (weak) explicit communication among the robots.

Acknowledgments. We would like to thank all people that have offered their ideas, comments, suggestions, and (conflicting) conjectures on this problem over the years. Especially, we would like to thank Elmo Welzl and Peter Widmayer.

References

1. L. Anderegg, M. Cieliebak, and G. Prencipe. A Linear Time Algorithm to Identify Equiangular and Biangular Point Configurations. Technical Report TR-03-01, Università di Pisa, 2003.
2. 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.
3. C. Bajaj. The Algebraic Degree of Geometric Optimization Problems. *Discrete and Computational Geometry*, 3:177–191, 1988.
4. T. Balch and R. C. Arkin. Behavior-based Formation Control for Multi-robot Teams. *IEEE Transaction on Robotics and Automation*, 14(6), 1998.
5. M. Cieliebak and G. Prencipe. Gathering Autonomous Mobile Robots. In *SIROCCO 9*, pages 57–72, 2002.
6. E. J. Cockayne and Z. A. Melzak. Euclidean Constructibility in Graph-minimization Problems. *Mathematical Magazine*, 42:206–208, 1969.
7. 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 *ISAAC '99*, volume 1741 of *LNCS*, pages 93–102, 1999.
8. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Autonomous Mobile Robots With Limited Visibility. In *STACS 2001*, volume 2010 of *LNCS*, pages 247–258, 2001.
9. D. Jung, G. Cheng, and A. Zelinsky. Experiments in Realising Cooperation between Autonomous Mobile Robots. In *ISER*, 1997.

10. Y. Kupitz and H. Martini. Geometric Aspects of the Generalized Fermat-Torricelli Problem. *Intuitive Geometry*, 6:55–127, 1997.
11. 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.
12. G. Prencipe. CORDA: Distributed Coordination of a Set of Autonomous Mobile Robots. In *ERSADS 2001*, pages 185–190, 2001.
13. G. Prencipe. Instantaneous Actions vs. Full Asynchronicity: Controlling and Coordinating a Set of Autonomous Mobile Robots. In *ICTCS 2001*, pages 185–190, 2001.
14. I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *Siam Journal of Computing*, 28(4):1347–1363, 1999.
15. E. Weiszfeld. Sur le Point Pour Lequel la Somme Des Distances de n Points Donnés Est Minimum. *Tohoku Mathematical*, 43:355–386, 1936.
16. E. Welzl. Smallest Enclosing Disks (Balls and Ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, LNCS, pages 359–370. Springer, 1991.