Meeting in a Polygon by Anonymous Oblivious Robots*

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

1 **University of Ottawa, Ottawa, Canada**
gdiluna@uottawa.ca
2 **University of Ottawa, Ottawa, Canada**
paola.flocchini@uottawa.ca
3 **Carleton University, Ottawa, Canada**
santoro@scs.carleton.ca
4 **University of Ottawa, Ottawa, Canada**
gvigliet@uottawa.ca
5 **Kyushu University, Fukuoka, Japan**
mak@inf.kyushu-u.ac.jp

Abstract

The *Meeting problem* for $k \geq 2$ searchers in a polygon P (possibly with holes) consists in making the searchers move within P, according to a distributed algorithm, in such a way that at least two of them eventually come to see each other, regardless of their initial positions. The polygon is initially unknown to the searchers, and its edges obstruct both movement and vision. Depending on the shape of P, we minimize the number of searchers k for which the Meeting problem is solvable. Specifically, if P has a rotational symmetry of order σ (where $\sigma = 1$ corresponds to no rotational symmetry), we prove that $k = \sigma + 1$ searchers are sufficient, and the bound is tight. Furthermore, we give an improved algorithm that optimally solves the Meeting problem with $k = 2$ searchers in all polygons whose barycenter is not in a hole (which includes the polygons with no holes). Our algorithms can be implemented in a variety of standard models of mobile robots operating in Look-Compute-Move cycles. For instance, if the searchers have memory but are anonymous, asynchronous, and have no agreement on a coordinate system or a notion of clockwise direction, then our algorithms work even if the initial memory contents of the searchers are arbitrary and possibly misleading. Moreover, oblivious searchers can execute our algorithms as well, encoding information by carefully positioning themselves within the polygon. This code is computable with basic arithmetic operations (provided that the coordinates of the polygon's vertices are algebraic real numbers in some global coordinate system), and each searcher can geometrically construct its own destination point at each cycle using only a compass. We stress that such memoryless searchers may be located anywhere in the polygon when the execution begins, and hence the information they initially encode is arbitrary. Our algorithms use a self-stabilizing map construction subroutine which is of independent interest.

1998 ACM Subject Classification I.2.11 Distributed Artificial Intelligence – multiagent systems

Keywords and phrases Meeting problem, Oblivious robots, Polygon, Self-stabilization

Digital Object Identifier 10.4230/LIPIcs.DISC.2017.14

* Full version available at https://arxiv.org/abs/1705.00324.

31st International Symposium on Distributed Computing (DISC 2017).
Editor: Andréa W. Richa; Article No. 14; pp. 14:1–14:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 Framework

Consider a set of $k \geq 2$ autonomous mobile robots, modeled as geometric points located in a polygonal enclosure P, which may contain holes. The boundary of P limits both visibility and mobility, in that robots cannot move or see through the edges of P. Each robot observes the visible portion of P (taking an instantaneous snapshot of it), executes an algorithm to compute a visible destination point, and then moves to that point. Such a *Look-Compute-Move cycle* is repeated forever by every robot, each time taking a new snapshot and moving to a newly computed point. In this paper we study the *Meeting problem*, which prescribes the k robots to move in such a way that eventually at least two of them come to see each other and become "mutually aware". We will refer to these robots as *P-searchers*, or simply *searchers*.

Our searchers are severely limited, which makes the Meeting problem harder to solve. They do not know the shape of P in advance, nor their whereabouts within P. They are anonymous, implying that they all execute the same algorithm to determine their destination points. They are oblivious, meaning that each destination point is computed based only on the last snapshot taken, while older snapshots are forgotten, and no memory is retained between cycles. They are deterministic, meaning that they cannot resort to randomness in their computations. They are asynchronous, in the sense that we make no assumptions on how fast each searcher completes a Look-Compute-Move cycle compared to the others (these parameters are dynamic and are controlled by an adversarial *scheduler*). They are disoriented, which means that they have no magnetic compasses, GPS devices, or agreements of any kind. Each searcher has its own independent local orientation, unit of length, and handedness. They are silent, in that they cannot communicate with one another in any way. They have arbitrary initial locations within P.

In real-life applications, being in line of sight may allow robots to communicate in environments where non-optical means of communication are unavailable or impractical [22]. Solving the Meeting problem is also a necessary preliminary step to more complex tasks, such as space coverage [23] or the extensively studied *Gathering problem*, where all k robots have to physically reach the same point and stop there. In the special case of $k = 2$ robots, the Gathering problem is also called *Rendezvous problem*. Clearly, the terminating condition of the Meeting problem is more relaxed than that of Gathering; hence, any solution to the Gathering problem would also solve Meeting. Unfortunately, no solution to the Gathering problem in the setting considered here exists in the literature (see Section 1.4), and to the best of our knowledge there are no previous results on the Meeting problem.

In fact, given our searchers' many handicaps, and especially their lack of memory and orientation, it is hard to see how they could solve any non-trivial problem at all. Nonetheless, in this paper we will present the surprising result that the Meeting problem is solvable in almost every polygon, even for $k = 2$ searchers. Moreover, with the addition of a simple synchronization phase, our Meeting algorithms can be turned into Rendezvous algorithms, as we will discuss in Section 5.

1.2 Techniques

Since our searchers are disoriented and have no kind of *a-priori* agreement, they must use the geometric features of P to implicitly agree on some "landmarks" which can help them in their task. In order to identify such landmarks, each searcher has to visit P and construct a

map of it. But this cannot be done straightforwardly, because searchers are oblivious, and they forget everything as soon as they move. To cope with this handicap, they carefully move within P in such a way as to implicitly encode information as their distance from the closest vertex.

This positional encoding technique poses some obvious difficulties. First, it greatly limits the freedom of the searchers: they have to do precise movements to encode the correct information, and still manage to visit all of P and update the map as they go. Second, since searchers can be located anywhere in P when the execution starts, they could be implicitly encoding anything. This includes misleading information, such as a false map of P that happens to be locally coherent with the surroundings of the searcher. Therefore, a searcher can never rely on the information it is implicitly encoding, but it must constantly re-visit the entire polygon to make sure that the map it is encoding is correct.

Hence, searchers cannot simply agree on a landmark and sit on it waiting for one another, because that would prevent them from re-visiting P. This inconvenience drastically complicates the Meeting problem, and forces the searchers to follow relatively complicated movement patterns that make at least two of them necessarily meet.

There is also a subtle problem with the actual encoding of complex data as the distance from a point, which is a single real number. One could naively pack several real numbers into one by interleaving their digits, but this encoding would not be computable by real random-access machines [3]. Hence, we propose a more sophisticated technique, which only requires basic arithmetic operations. Such a technique can substitute the naive one under the reasonable assumption that the vertices of P be points with algebraic coordinates (as expressed in some global coordinate system, which is not necessarily the local one of any searcher).

1.3 Our Contributions

We prove that the Meeting problem in a polygon P can be solved by $k = \sigma + 1$ searchers, where σ is the order of the rotation group of P (which is also called the *symmetricity* of P). We also give a matching lower bound, showing that there are polygons of symmetricty σ where σ searchers cannot solve the Meeting problem.

Then, since all our lower-bound examples are polygons with a hole around the center, we wonder if the Meeting problem can be solved by fewer searchers if we exclude this small class of polygons. Surprisingly, it turns out that in all the remaining polygons only two searchers are sufficient to solve the Meeting problem. In particular, these include all the polygons with no holes.

Additionally, searchers can geometrically construct their destination points with a compass, provided that the vertices of P are algebraic points. Equivalently, searchers only have to compute combinations of basic arithmetic operations and square root extractions on the coordinates of the visible vertices of P. This is done via an encoding technique of independent interest, which we apply to mobile robots for the first time.

As a subroutine of our algorithms, we employ a self-stabilizing map construction algorithm that is of independent interest, as well.

In Section 2, we formally define all the elements of the Meeting problem. In Section 3, we consider the Meeting problem for searchers equipped with an unlimited amount of persistent internal memory whose initial contents can be arbitrary (hence also "incorrect"). This simplification allows us to present "cleaner" versions of our algorithms, which are not burdened by the technicalities of our positional encoding method. In Section 4, we present our encoding technique and we show how to apply it to the Meeting algorithms of Section 3, thus

extending our results to oblivious searchers. Finally, in Section 5 we discuss complementary results and directions for further research. Among other things, we briefly explain how to convert our Meeting algorithms into Rendezvous algorithms.

1.4 Related Work

The Gathering problem has been extensively studied in several contexts [1, 16]. The literature can be divided into works considering robots in a geometric space, and works considering agents on a graph [1, 11, 14]. The works on Gathering in the plane, which are more related to our setting, pertain to robots that inhabit an unbounded plane where no extraneous objects can block visibility or movement [2, 7, 12, 15, 18, 19]. In particular, none of these results considers robots in a polygon.

The work that is most relevant to ours is represented by a series of papers on Rendezvous and approximate Rendezvous by two robots in polygons or more general planar enclosures [9, 10, 11, 13]. The authors show how to guarantee that the two robots' trajectories will intersect (or get arbitrary close to each other in case of approximate rendezvous) within finite time, in spite of a powerful adversary that controls the speed and the movements of the robots on their trajectories. However, termination happens implicitly: the robots are not necessarily aware of each other's presence, and Rendezvous is considered solved even if they are both moving. Moreover, none of these papers considers oblivious robots, and none of them allows the initial memory contents of the robots to be arbitrary. Both are simplifications of the problem, because they allow robots to implicitly agree on a single landmark and just move there.

Other mildly related works consider robots searching for an intruder in a polygon [27, 28], robots in an empty plane that obstruct each other's view, whose task is to become all mutually visible [20, 24, 25], robots tasked with constructing a map of the visibility graph of a polygon [4, 5, 6], and a static version of the Meeting problem called the *hidden set problem* [26].

The issue of defining a model of computation for mobile robots has hardly ever been addressed in the relevant literature. It is nonetheless interesting to establish what destination points are computable by mobile robots, and what it means for them to compute a point. To the best of our knowledge, only two papers deal with this problem [7, 17], explicitly rejecting transcendental functions and deeming them not intuitively computable. Interestingly, other papers, such as [9], allow robots to compute transcendental functions.

Another contribution of this paper is a formal definition of the concept of computability for mobile robots (see the beginning of Section 4.2). Accordingly, all of our geometric constructions can be performed with a compass.

2 Definitions

A *polygon* in the Euclidean plane $\mathbb{R}^2$ is a non-empty, bounded, connected, and topologically closed 2-manifold whose boundary is a finite collection of line segments. The *vertices*, *edges*, and *diagonals* of a polygon are defined in the standard way, as well as the notion of *adjacency* between vertices. One connected component of a polygon's boundary, called the *external boundary*, encloses all others, which are called *holes*. We say that a point $p \in P$ *sees* a point $q \in P$ (or, equivalently, that q is *visible* to p) if the line segment pq lies in P.

If a polygon has an axis of symmetry, we say that it is *axially symmetric*. The largest integer σ such that rotating a polygon around its barycenter by $2\pi/\sigma$ radians leaves it unchanged is called the *symmetricity* of the polygon. In other words, the symmetricity is the order of the rotation group of the polygon. If $\sigma > 1$, the polygon is said to be *rotationally symmetric*.

If P is a polygon, by *P-searcher* we mean an anonymous *robot* represented by a point in P, which, informally, can *observe* its surroundings and *move* within P. The technical specifications of our searchers have been listed in Section 1.1. We remark that each searcher's snapshots are expressed in its own *local reference system*, which is a Cartesian system of coordinates with the searcher's current location as the origin. A searcher's local coordinate system translates as the searcher moves (to keep the searcher's location at the origin), but it retains its orientation, scale, and handedness.

We say that two P-searchers are *mutually aware* at some point in time if they have seen each other during their most recent Look phases. That is, if searcher s_1 sees searcher s_2 during a Look phase at time t_1, s_2 sees s_1 during a Look phase at time $t_2 \geq t_1$, and neither s_1 nor s_2 performs another Look phase in the time interval (t_1, t_2), then s_1 and s_2 are mutually aware at time t_2 (and they remain mutually aware until s_1 performs a Look phase without seeing s_2, or vice versa). A very similar notion of mutual awareness has been defined in [21].

Given a team of P-searchers, the *Meeting problem* prescribes that at least two of them become mutually aware. More precisely, the Meeting problem for k searchers in P is *solvable* if there exists an algorithm A such that, if all k searchers execute A during all their Compute phases, at least two of them eventually become mutually aware, regardless of how the searchers are initially laid out in P, and regardless of how the scheduler decides to control their behavior. Occasionally, we will say that two searchers *meet*, as a synonym of becoming mutually aware.

In Section 3, we are going to assume that each searcher has an unlimited amount of *persistent internal memory*, which can be read and updated by the searcher during each Compute phase, and is retained for use in later Compute phases. The initial contents of the internal memory of each searcher are arbitrary, and possibly "incorrect". In Section 4, we will drop the persistent memory requirements, and we will extend our algorithms to *oblivious* searchers, whose computations only rely on the single snapshot taken in the most recent Look phase, and whose internal memory is erased during each Move phase.

3 Algorithms and Correctness

3.1 General Algorithm

First we give a lower bound on the minimum number of searchers required to solve the Meeting problem in a polygon. Our bound is in terms of the polygon's symmetricity.

▶ **Theorem 1.** *For every integer $\sigma > 0$, there exists a polygon with symmetricity σ in which σ (or fewer) searchers cannot solve the Meeting problem.*

Proof. If $\sigma = 1$, the statement is trivial. If $\sigma > 1$, we construct a polygon with symmetricity σ shaped as a σ-pointed star with one large hole almost touching the external boundary, as shown in Figure 1. We then arrange $\sigma' \leq \sigma$ searchers and orient their local coordinate systems in a symmetric fashion, as in Figure 1. Now, let the initial memory contents of all the searchers be equal, and suppose that the scheduler always activates them synchronously. By the rotational symmetry of our construction, each searcher gets an identical snapshot of the polygon, and therefore all searchers compute symmetric destination points and modify their memory in the same way. This holds true at every cycle, and so, by induction, the searchers will always be found at σ' symmetric locations throughout the execution. Note that our polygon has the property that no two of its points whose angular distance (with respect to the barycenter) is a multiple of $2\pi/\sigma$ can see each other. Hence, no matter what algorithm the searchers are executing, no two of them will ever be mutually aware. ◀

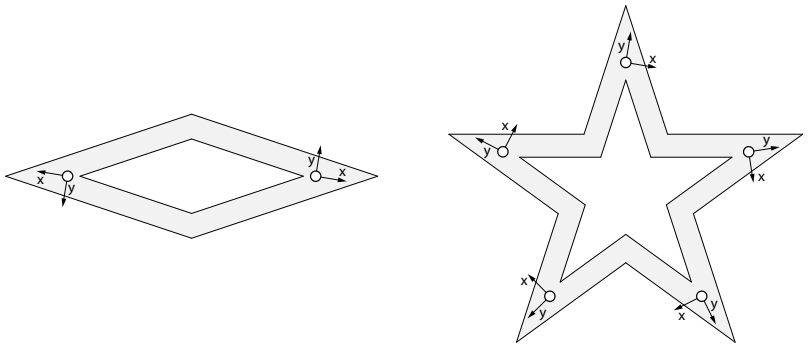

Figure 1 Constructions used in Theorem 1 for $\sigma = 2$ and $\sigma = 5$.

Algorithm 1 Meeting algorithm for general polygons.

```
Persistent variables
SnapshotList
Action
Direction
Polygon
PivotPoint

Procedure Compute (Snapshot)
if Snapshot contains no other searcher then
    Append Snapshot to SnapshotList
    if SnapshotList is inconsistent or (Action = PATROL and PivotPoint is not consistent with Polygon) then
        SnapshotList := Snapshot
        Action := EXPLORE
    if Action = EXPLORE then
        U := Extract unvisited vertices from SnapshotList
        if U ≠ ∅ then
            v := First vertex of U
            Move to the next vertex in a shortest path to v
        else
            Action := PATROL
            Direction := CLOCKWISE
            Polygon := Extract polygon from SnapshotList
            S := Set of axes of symmetry of Polygon
            if S = ∅ then
                C := Select a rotation class of vertices of Polygon in a similarity-invariant way
                PivotPoint := Select any vertex in C
            else
                S' := Select a class of equivalent axes in S in a similarity-invariant way
                ℓ := Select any axis in S'
                C := Select a class of points of ℓ on the boundary of Polygon in a similarity-invariant way
                PivotPoint := Select any point in C
            Augment Polygon using PivotPoint as pivot in a similarity-invariant way to make it simply connected
    if Action = PATROL then
        if I am in PivotPoint then
            Invert Direction
        Move to the next vertex of Polygon, following its boundary in the direction stored in variable Direction
```

Next we will prove that the bound of Theorem 1 is tight, and hence $\sigma + 1$ searchers are optimal. We do so by giving a Meeting algorithm called Algorithm 1, which works by constructing a map of the polygon in a self-stabilizing way. This algorithm assumes that searchers have unlimited memory, and hence they can store the entire history of the snapshots they have taken since the beginning of the execution. In Section 4, we will show how to drop this requirement and apply our algorithms to oblivious searchers.

By definition, the Meeting problem is solved when two searchers become mutually aware. So, in our algorithm, whenever a searcher sees another searcher, it stays idle for a cycle and waits to be noticed (which does not necessarily happen, since the second searcher may be in the middle of a Move phase and disappear shortly after). Let P be the polygon in which

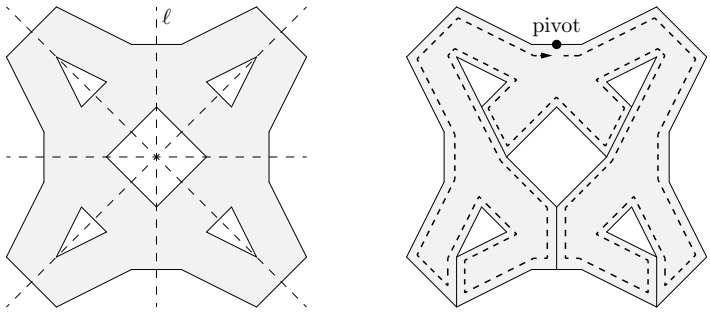

Figure 2 Augmenting an axially symmetric polygon and defining a tour of its boundary.

the searchers are located. Since the initial memory contents may be incorrect, if a searcher notices a discrepancy between the current snapshot of P and the history of snapshots stored in memory, it forgets everything and restarts the execution from wherever it is.

As observed in Section 1.2, each searcher must keep re-visiting every part of the boundary of P. Hence, our main algorithm is divided into two phases: EXPLORE and PATROL. Roughly speaking, in the EXPLORE phase, a searcher visits all vertices of P; in the PATROL phase, it moves back and forth along the boundary of P, searching for a companion. The EXPLORE phase is relatively simple: as the searcher explores new vertices, it keeps track of the ones that it has seen but not visited. Then it picks the first of such vertices and moves to it along a shortest path.

For the PATROL phase, the searcher must first choose a *pivot point* of P, which is the point where the searcher changes direction as it patrols P's boundary. It also has to cope with the fact that the boundary of P may not be connected, since P may have holes. The pivot point is always chosen on the boundary of P and on a symmetry axis of P, if one exists. It is also chosen in a *similarity-invariant* way, meaning that the selection algorithm should not depend on the scale, rotation, position, and handedness of P, but it should be a deterministic algorithm that only looks at angles between vertices and ratios between segment lengths. This is to guarantee that all searchers that have a correct picture of P in memory (expressed in their respective local coordinates systems) will select pivot points that are equivalent up to similarity.

Once a searcher has selected a pivot point, it adds some "artificial" edges to P in order to make it simply connected, i.e., remove all its holes. This may be impossible to do in a similarity-invariant way, so the pivot point is used to determine how symmetries are broken. An example of this construction is illustrated in Figure 2, where the polygon is symmetric and therefore the additional edges are symmetric, as well.

As a result of cutting P along these segments, we obtain a *degenerate* simply connected polygon $\widetilde{P}$. It is now possible to perform a *tour* of the boundary of $\widetilde{P}$, as shown in Figure 2. The PATROL phase of our algorithm consists in taking a tour of $\widetilde{P}$ and switching direction (from clockwise to counterclockwise and vice versa) every time the pivot point is reached. So, all vertices of P are perpetually visited in some fixed order, then in the opposite order, and so on. At any time, the searcher can always determine its next destination vertex based on the history of snapshots stored in memory.

▶ **Theorem 2.** *There is an algorithm that, for every integer $\sigma > 0$, solves the Meeting problem with $\sigma + 1$ searchers (regardless of their initial memory contents) in every polygon with symmetricity σ.*

Proof. We will show that Algorithm 1 correctly solves the Meeting problem for $\sigma+1$ searchers in any polygon P with symmetricity σ. Since the initial memory contents of a searcher may be incorrect, when a searcher notices a discrepancy between the current observation and a previous observation, it erases its own memory and restarts the execution. The same happens if it realizes that the pivot point it has chosen does not match the polygon. From that point onward, the searcher's memory will only contain correct information, and the execution will never be restarted again. Hence, in the following, we will assume that no such discrepancy is ever discovered, and therefore the execution is never restarted.

In the EXPLORE phase, a searcher moves to all the discovered but not yet visited vertices of P, until there are none left. This indeed lets the searcher discover all vertices of P, due to the connectedness of its *visibility graph* (which is the graph on the set of vertices of P whose edges are the edges and diagonals of P). This means that eventually the searcher will have a complete and coherent representation of *some* polygon in memory. If this representation is incorrect, the searcher will eventually find out during the PATROL phase, when it re-visits all vertices of P.

We can therefore assume without loss of generality that, at some point, all searchers are in the PATROL phase, and all have a correct representation of P in memory. Since the pivot point is chosen by each searcher in a similarity-invariant way, there are only σ points that can possibly be elected as pivots. By the pigeonhole principle, there are two searchers that have the same pivot point. So, both searchers will perform a clockwise tour of the boundary of $\widetilde{P}$, touching all of its vertices is some fixed order, followed by a counterclockwise tour, touching all vertices in the opposite order, and so on. Since they both turn around at the same pivot point, they do the same tour. As a consequence, they become mutually aware by the time one of them has completed a full tour, thus solving the Meeting problem. ◀

We emphasize that, if a searcher were tasked to construct a map of P, it could do so by simply executing the above algorithm indefinitely (i.e., ignoring the presence of other searchers). Since the algorithm eventually discovers and corrects any possible inconsistency in the initial memory state of the searcher, it is self-stabilizing.

3.2 Improved Algorithm for Polygons with Barycenter not in a Hole

The worst-case examples given in Theorem 1 are polygons with a hole around the barycenter. It is natural to wonder if the Meeting problem can be solved with fewer searchers if we exclude this special type of polygons. It turns out that in all other cases Algorithm 1 can be drastically improved: only two searchers are needed whenever the polygon's barycenter is not in a hole. Notably, this includes all polygons with no holes.

It is not hard to construct counterexamples where simply making the searchers patrol the boundary of the polygon as in the previous algorithm may not solve the Meeting problem, even if the polygon has no holes. Hence a new strategy has to be devised: our improved Meeting algorithm is called Algorithm 2. It begins by testing for the presence of another searcher, followed by some consistency tests, and an EXPLORE phase, which are essentially the same as in the previous algorithm. It then proceeds with a PATROL phase, which is more complex than the old one. Note that Algorithm 1 already solves the Meeting problem with two searchers if the polygon is not rotationally symmetric (i.e., for $\sigma = 1$). So, in this special case, our improved algorithm works exactly as the previous one. In the following, we will therefore assume that the polygon is rotationally symmetric, and we will discuss only the new PATROL phase.

Algorithm 2 Improved Meeting algorithm for polygons with barycenter not in a hole.

```
Persistent variables
SnapshotList
Action
Stage
Polygon
PivotVertex
PolygonTriangles
PolygonLevels

Procedure Compute (Snapshot)
if Snapshot contains no other searcher then
    Append Snapshot to SnapshotList
    if Persistent variables are inconsistent then
        SnapshotList := Snapshot
        Action := EXPLORE
    if Action = EXPLORE then
        U := Extract unvisited vertices from SnapshotList
        if U ≠ ∅ then
            v := First vertex of U
            Move to the next vertex in a shortest path to v
        else
            Action := PATROL
            Stage := −1
            Polygon := Extract polygon from SnapshotList
            if Polygon is rotationally symmetric then
                C := Select a class of vertices of Polygon closest to the center in a similarity-invariant way
                PivotVertex := Select any vertex in C
                Augment Polygon in a similarity-invariant way to make it simply connected
                Triangulate each branch of augmented Polygon in a similarity-invariant way
                PolygonTriangles := Total number of triangles in the triangulation of augmented Polygon
                PolygonLevels := Height of the dual tree of the triangulation of each branch of augmented Polygon
            else
                PivotVertex := Select a vertex of Polygon in a similarity-invariant way
    if Action = PATROL then
        if Polygon is rotationally symmetric then
            if I am in PivotVertex then
                Stage := Stage + 1
                if Stage ≥ 2 · PolygonLevels + 2 · PolygonTriangles^2 then
                    Stage := 0
            if Stage = −1 then
                Move to the next vertex in a shortest path to PivotVertex
            else if Stage < PolygonLevels then
                j := Stage
                Move to the next vertex of a clockwise j-tour of Polygon
            else
                j := 2 · PolygonLevels + 2 · PolygonTriangles^2 − Stage
                if j > PolygonLevels then
                    j := PolygonLevels
                Move to the next vertex of a counterclockwise j-tour of Polygon
        else
            if I am in PivotVertex then
                Stage := Stage + 1
            if Stage is odd then
                Move to the next vertex of Polygon, following its boundary in the clockwise direction
            else
                Move to the next vertex of Polygon, following its boundary in the counterclockwise direction
```

Upon ending the EXPLORE phase, a searcher does some pre-processing on the polygon P. First it identifies a polygon Q formed by vertices of P that are closest to the center of rotation and equivalent up to similarity. So, Q is a convex polygon that is completely contained in P. Each connected component of $P \setminus Q$ is called a *branch* of P. Each branch is then augmented by some extra edges to make it simply connected, obtaining a degenerate polygon $\widetilde{P}$. Then, $\widetilde{P} \setminus Q$ is triangulated in a similarity-invariant way. This guarantees that all searchers compute the same triangulation. Figure 3 shows an example of this construction.

Let P_j be the union of Q and the triangles of the triangulation whose corresponding nodes of the dual graph have distance at most j from the root Q. Let m be the smallest integer such that $P_m = P$. The PATROL phase has a "primitive" operation called j-*tour*,

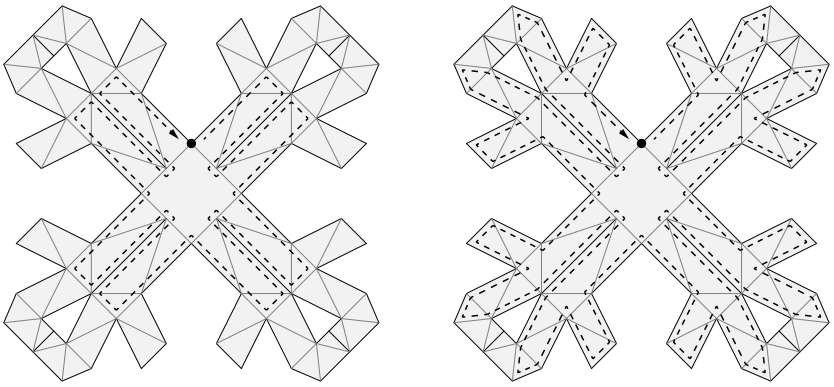

Figure 3 Augmented and triangulated axially symmetric polygon with a 3-tour and a 6-tour.

with $0 \leq j \leq m$, which is a tour of the boundary of P_j, starting and ending at the pivot point, following the edges of $\widetilde{P}$. For example, a 0-tour is simply a tour of the boundary of Q, an m-tour is a tour of the boundary of $\widetilde{P}$ (much like the tours of Algorithm 1), and Figure 3 shows a 3-tour and a 6-tour.

The PATROL phase consists of several *stages*, and in each stage the searcher performs a j-tour, for some j. The j-tours are performed according to the following list, which is repeated until the Meeting problem is solved: a clockwise 0-tour, a clockwise 1-tour, a clockwise 2-tour, ..., a clockwise $(m-1)$-tour, a sufficiently large number of counterclockwise m-tours (twice the square of the total number of triangles in the triangulations of all the branches of P is abundantly enough), a counterclockwise $(m-1)$-tour, a counterclockwise $(m-2)$-tour, ..., a counterclockwise 1-tour. The first m stages, where the searcher performs clockwise j-tours, are called *ascending stages*. All the other stages are called *descending stages*. Moreover, the first stage is called the *central stage*, and the stages in which an m-tour is performed are called *perimeter stages*.

Note that, when we say "clockwise" and "counterclockwise", we mean it in the local reference system of the executing searcher. Recall that two different searchers executing the algorithm may not have the same notion of clockwise direction, and therefore in their respective ascending stages they may actually perform tours in opposite directions. If two searchers have the same notion of clockwise direction, they are said to be *concordant*; otherwise, they are *discordant*.

We can now proceed with the proof of correctness of Algorithm 2. The proof of the following technical lemmas is found in the appendix.

▶ **Lemma 3.** *Let two P-searchers be executing Algorithm 2, let both be in the PATROL phase, and let both have a correct representation of the polygon P in memory, which is rotationally symmetric. Then, they will either become mutually aware or be in a perimeter stage at the same time, with at least one full perimeter stage still to perform.*

▶ **Lemma 4.** *Assume the hypotheses of Lemma 3, and let the searchers be concordant. If one searcher begins a j-tour in an ascending (respectively, descending) stage while the other searcher is performing a $(j+1)$-tour (respectively, $(j-1)$-tour) in a descending (respectively, ascending) stage, with $0 \leq j < m$ (respectively, $0 < j < m$), they eventually become mutually aware.*

▶ **Theorem 5.** *There is an algorithm that solves the Meeting problem with two searchers (regardless of their initial memory contents) in every polygon whose barycenter does not lie in a hole.*

Proof. We will show that Algorithm 2 correctly solves the Meeting problem for two searchers in any polygon P whose barycenter does not lie in a hole. The proof of correctness is the same as that of Theorem 2, except for the PATROL phase. So, in the following, we will assume that both searchers already have a correct picture of P in memory, and are both in the PATROL phase. Moreover, since the new algorithm works in the same way as the old one if P is not rotationally symmetric (and the proof of correctness is the same as in Theorem 2), we will assume that P is rotationally symmetric.

Suppose for a contradiction that the searchers never become mutually aware. By Lemma 3, at some point they are both in a perimeter stage. If they are discordant, they proceed along the boundary of $\widetilde{P}$ in opposite directions, and therefore they become mutually aware. If they are concordant, they will perform all the remaining descending stages, followed by the ascending stages, starting with the central stage. If they start the central stage at the same time, they necessarily become mutually aware, because they are on the boundary of the central polygon Q, which is convex and empty. So, one searcher must begin the central stage while the other is still in a descending stage. Then, as one searcher ascends and the other descends, the hypotheses of Lemma 4 are going to be satisfied, which means that the searchers eventually become mutually aware. ◀

4 Memoryless Implementations

4.1 Encoding Persistent Variables

In order to re-implement our algorithms without using any memory, we first want to be able to encode all persistent variables (which roughly amount to a list of snapshots) as a single real number. Since a snapshot is the visible sub-polygon of P expressed in the coordinate system of the observing P-searcher, it can be easily expressed as the array of the visible sub-segments of P's edges, or as the array of the coordinates of these segments' endpoints. Hence, encoding a history of snapshots amounts to encoding a finite array of real numbers $(r_1, \ldots, r_n)$ as a single real number r.

As a model of computation, we choose the *Blum-Shub-Smale machine* [3], which is a random-access machine whose registers can store arbitrary real numbers. Its computational primitives are the four basic arithmetic operations, but it is customary to extend the basic model with additional primitives, which should be somewhat well-behaved and intuitively computable. This immediately rules out the naive approach of constructing the binary representation of r by interleaving the binary digits of the r_i's. Indeed, such an encoding function has a set of discontinuities that is dense in its entire domain, and is therefore not intuitively computable.

We propose a more sophisticated encoding strategy, which is computable even on a basic Blum-Shub-Smale machine (i.e., the one with the four basic arithmetic operations only). A small drawback is that we can only apply this method if the vertices of the polygon P have algebraic coordinates (i.e., they are *algebraic points*) in some global coordinate system. In practice, we are not imposing a big limitation on our inputs, in that basically all the polygons we can reasonably think of fall into this class. In particular, we can construct polygons of any symmetricity (by contrast, using rational points would only allow us to construct polygons with symmetricity 1, 2, and 4).

First of all, we observe that we can represent an algebraic number using finitely many bits: if the algebraic number α is the ith real root of the polynomial with integer coefficients $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, we can encode it as $(n, i, a_n, a_{n-1}, \ldots, a_0)$, which in turn requires only finitely many bits. Given α, a suitable polynomial can be found by simply generating all possible polynomials in some order (they are a countable set) and evaluating them in α.

Once we have some algebraic numbers expressed in this implicit finite form, we can use our basic Blum-Shub-Smale machine to do Turing-computable bit manipulations to compute all kinds of common functions on them. In particular, there are standard ways of computing the basic arithmetic operations, as well as root extractions of any degree. A comprehensive exposition of these techniques, along with their theoretical background, is found in [8]. Essentially, this is also one of the several ways in which mathematical software such as Sage, Mathematica, and CGAL handles algebraic numbers and does exact computations with then.

The key point to keep in mind is that, once a number is encoded in this form, we cannot necessarily retrieve it in finite time; we can only approximate it arbitrarily well, for instance via Sturm's theorem [8]. However, we can still evaluate computable predicates on these numbers exactly, and have them influence the flow of our algorithms.

Note that, even if the vertices of P are algebraic points in some global coordinate system, they may not be algebraic in the coordinate system of a searcher. So, the idea is to let the searcher use two visible vertices v and v' as a basis to construct a new coordinate system, and then compute the coordinates of all other visible vertices of P in this system. The resulting coordinates are guaranteed to be algebraic numbers, and can be encoded with the above technique. Note that some vertices of the snapshot are not necessarily vertices of P, and may not be algebraic even in the new coordinate system. These vertices are easily identified and discarded. This causes some loss of information, which is not going to matter in our application to the Meeting problem.

4.2 Adapting the Algorithms

Next we are going to apply our encoding method to the Meeting algorithms of Section 3, and we will show how oblivious searchers can solve the Meeting problem, as well. We will need searchers to be able to compute only basic arithmetic operations on real numbers, as well as extract square roots. Hence, internally, each searcher will run a Blum-Shub-Smale machine extended with a square-root primitive. Only the four basic arithmetic operations are required for our encoding method, but square roots are needed in the geometric computations. It is well known that the points whose coordinates can be computed by composing these five operations are precisely the ones that can be constructed with a compass and a straightedge. In turn, the Mohr-Mascheroni theorem states that these points can also be constructed with a compass alone.

A searcher can simulate memory by encoding data as its distance d from the closest visible vertex of P, which we call the *virtual vertex* of the searcher. It is not hard to design the code in such a way that d and $d/2$ have the same meaning, for every d. This gives a searcher the ability to get arbitrarily close to its virtual vertex without losing memory. After decoding d, a searcher will execute one of the old Meeting algorithms pretending to be located on the virtual vertex. This approach poses several technical problems, which we examine next.

Recall that snapshots are encoded in a special coordinate system defined by two visible vertices v and v'. As the searcher moves to a different virtual vertex, it has to re-express all the history of snapshots into a new coordinate system, choosing a v and a v' that will be visible from its destination point. This can be done by letting v be the current virtual vertex and v' be the destination virtual vertex, and adding enough information to the encoded data to retrieve v once the virtual vertex is v'. This way, the searcher can transport snapshots around P and is still able to decode them.

In the EXPLORE phase, a searcher completely explores one connected component of the boundary of P before moving to the next component. To determine the position of the next vertex, it has to move close enough to the current virtual vertex, and possibly around it if it

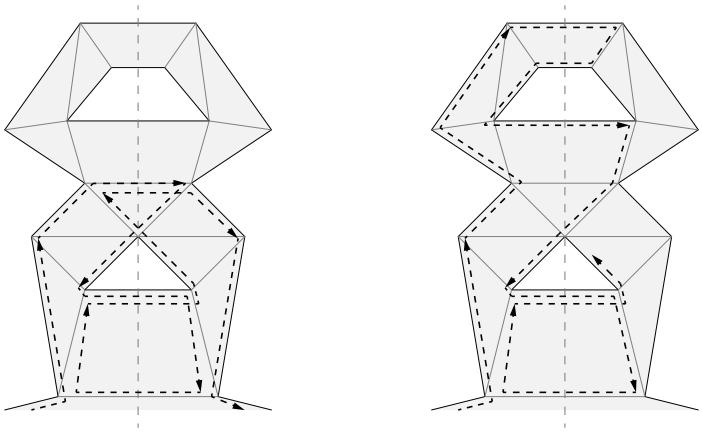

Figure 4 Symmetric partition of a branch with a 5-tour and part of a perimeter tour.

is reflex. The searcher makes sure to stop on the angle bisector of each vertex it visits and take a snapshot from there. If implemented properly, this strategy guarantees that, when the list of discovered but not visited vertices is empty, all vertices of P have actually been visited, and the map of P is faithful.

For the PATROL phase, recall that a tour turns at the vertices of the augmented polygon $\widetilde{P}$, which are not necessarily vertices of P. Unfortunately, our oblivious searchers cannot approach generic points without losing information. Our solution is to modify the tours so that they turn only at vertices of P. These modifications should retain the axes of symmetry of the tours whenever they are required by the original algorithms. This is especially tricky for the j-tours of Algorithm 2, where we need to redesign the partition of $\widetilde{P}$ including not only triangles but also isosceles trapezoids, as shown in Figure 4. Note that our new j-tours may self-overlap and touch the same vertices multiple times, but this is not an issue. These new j-tours have the relevant properties that are required by Lemmas 3 and 4: the pieces of the partition are convex, and each point on a $(j+1)$-tour is visible to at least an entire edge of a j-tour.

Of course, a j-tour defines a curve that is followed only approximately by our oblivious searchers, because they have to keep encoding information as they go. Still, they follow this curve closely enough and without ever properly crossing it, in such a way as to preserves the aforementioned relevant properties. Also, while they do so, they have to stop on the angle bisector of each vertex whenever they get the opportunity: this is to guarantee that they will discover any discrepancies between the encoded map and the real polygon.

Finally, if two searchers are doing the same tour in opposite directions, and they stay within a thin-enough "band" around the correct curve, they are guaranteed to become mutually aware when they cross each other.

Therefore, Theorems 2 and 5 can be extended to oblivious searchers.

▶ **Theorem 6.** *There is an algorithm that, for every integer $\sigma > 0$, solves the Meeting problem with $\sigma + 1$ oblivious searchers in every polygon with symmetricity σ. There is an algorithm that solves the Meeting problem with two oblivious searchers in every polygon whose barycenter does not lie in a hole. If the polygon's vertices are algebraic points, these algorithms are implementable on a real random-access machine that can compute basic arithmetic operations and extract square roots.*

5 Conclusions and Further Work

Our Meeting algorithms can be converted into Rendezvous algorithms for two searchers. Indeed, once two searchers realize that they have become mutually aware, they can implicitly agree on a rendezvous point using the boundary of the polygon as a static reference. Note that this "rendezvous phase" should only be performed if the searchers are in fact mutually aware, or else they may behave incorrectly and compromise the algorithms. To detect mutual awareness, a searcher that sees another searcher performs a preliminary "synchronization phase", in which it moves very slightly a few times and sees if the other searcher does the same or disappears. If the pattern of these "synchronization moves" is unique to this phase, i.e., it cannot be normally performed during an EXPLORE or a PATROL phase, then the searchers know that they are mutually aware, and they can proceed to the rendezvous phase.

In some cases, we can also extend our algorithms to weaker models of searchers. Namely, searchers with *limited visibility*, which can only see up to a fixed distance, and the *non-rigid* setting, in which a searcher can be stopped by the scheduler during each Move phase before reaching its destination point, but not before having moved by at least a constant δ (for details on this model, refer to [16]). However, a complete solution for these and other searcher models remains an open problem.

References

1 S. Alpern and S. Gal. *The theory of search games and rendezvous*. Springer, 2003.

2 H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 15(5):818–828, 1999.

3 L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation*. Springer-Verlag New York, 1998.

4 J. Chalopin, S. Das, Y. Disser, M. Mihalák, and P. Widmayer. Mapping simple polygons: how robots benefit from looking back. *Algorithmica*, 65(1):43–59, 2013.

5 J. Chalopin, S. Das, Y. Disser, M. Mihalák, and P. Widmayer. Simple agents learn to find their way: an introduction on mapping polygons. *Discrete Applied Mathematics*, 161(10–11):1287–1307, 2013.

6 J. Chalopin, S. Das, Y. Disser, M. Mihalák, and P. Widmayer. Mapping simple polygons: the power of telling convex from reflex. *ACM Transactions on Algorithms*, 11(4):33:1–33:16, 2015.

7 M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.

8 H. Cohen. *A course in computational algebraic number theory*. Springer, 1993.

9 J. Czyzowicz, D. Ilcinkas, A. Labourel, and A. Pelc. Asynchronous deterministic rendezvous in bounded terrains. *Theoretical Computer Science*, 412(50):6926–6937, 2011.

10 J. Czyzowicz, A. Kosowski, and A. Pelc. Deterministic rendezvous of asynchronous bounded-memory agents in polygonal terrains. *Theory of Computing Systems*, 52(2):179–199, 2013.

11 J. Czyzowicz, A. Labourel, and A. Pelc. How to meet asynchronously (almost) everywhere. *ACM Transactions on Algorithms*, 8(4):37:1–37:14, 2012.

12 X. Défago, M. Gradinariu, S. Messika, and P. Raïpin-Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering. In *Proceedings of the 20th International Conference on Distributed Computing, (DISC)*, pages 46–60, 2006.

13 Y. Dieudonné and A. Pelc. Deterministic polynomial approach in the plane. *Distributed Computing*, 28(2):111–129, 2015.

14 Y. Dieudonné, A. Pelc, and V. Villain. How to meet asynchronously at polynomial cost. *SIAM Journal on Computing*, 44(3):844–867, 2015.

15 Y. Dieudonné and F. Petit. Self-stabilizing gathering with strong multiplicity detection. *Theoretical Computer Science*, 428:47–57, 2012.

16 P. Flocchini, G. Prencipe, and N. Santoro. *Distributed computing by oblivious mobile robots*. Morgan & Claypool, 2012.

17 P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta. Distributed computing by mobile robots: Uniform Circle Formation. *Distributed Computing*, in press.

18 P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1–3):147–168, 2005.

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

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

21 L. Pagli, G. Prencipe, and G. Viglietta. Getting close without touching: Near-Gathering for autonomous mobile robots. *Distributed Computing*, 28(5):333–349, 2015.

22 W. Rabinovich, J. Murphy, M. Suite, M. Ferraro, R. Mahon, P. Goetz, K. Hacker, W. Freeman, E. Saint Georges, S. Uecke, and J. Sender. Free-space optical data link to a small robot using modulating retroreflectors. In *Proceedings of SPIE 7464, Free-Space Laser Communications IX*, volume 7464, 2009.

23 I. Rekleitis, V. Lee-Shue, A. Peng New, and H. Choset. Limited communication, multi-robot team based coverage. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3462–3468, 2004.

24 G. Sharma, C. Busch, and S. Mukhopadhyay. Mutual visibility with an optimal number of colors. In *Proceedings of the 11th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*, pages 196–210, 2016.

25 G. Sharma, R. Vaidyanathan, J. L. Trahan, C. Busch, and S. Rai. Complete visibility for robots with lights in O(1) time. In *Proceedings of the 18th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 327–345, 2016.

26 T. Shermer. Hiding people in polygons. *Computing*, 42(2):109–131, 1989.

27 I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, 1992.

28 M. Yamashita, H. Umemoto, I. Suzuki, and T. Kameda. Searching for mobile intruders in a polygonal region by a group of mobile searchers. *Algorithmica*, 31(2):208–236, 2001.