

---

## Subhash Suri

Department of Computer Science  
University of California  
Santa Barbara, USA 95106

## Elias Vicari Peter Widmayer

Institute of Theoretical Computer Science  
ETH Zurich  
8092 Zurich, Switzerland  
vicariel@inf.ethz.ch

# Simple Robots with Minimal Sensing: From Local Visibility to Global Geometry

## Abstract

*We consider problems of geometric exploration and self-deployment for simple robots that can only sense the combinatorial (non-metric) features of their surroundings. Even with such a limited sensing, we show that robots can achieve complex geometric reasoning and perform many non-trivial tasks. Specifically, we show that one robot equipped with a single pebble can decide whether the workspace environment is a simply connected polygon and, if not, it can also count the number of holes in the environment. Highlighting the subtleties of our sensing model, we show that a robot can decide whether the environment is a convex polygon, yet it cannot resolve whether a given vertex is convex. Finally, we show that by using such local and minimal sensing a robot can compute a proper triangulation of a polygon and that the triangulation algorithm can be implemented collaboratively by a group of  $m$  such robots, each with  $\Theta(n/m)$  word memory. As a corollary of the triangulation algorithm, we derive a distributed analog of the well-known Art Gallery Theorem. A group of  $\lfloor n/3 \rfloor$  (bounded memory) robots in our minimal sensing model can self-deploy to achieve visibility coverage of an  $n$ -vertex art gallery (polygon). This resolves an open question raised recently.*

KEY WORDS—artificial intelligence, robots

## 1. Introduction

The study of simple robot systems, with minimalistic sensory input, is of fundamental interest in both theory and practice.

In theory, a minimalistic model provides a clean conceptual framework for performance analysis and lends insights into the inherent complexity of various tasks: the positive results identify the easy problems, while the negative results help isolate the difficult problems that necessitate richer functionality and sensing. Models with simple sensing are also robust against noise and help simplify the information (belief) space of the robot systems, whose complexity is often a source of much difficulty in planning (LaValle 2006). On the practical side, robots with a simple sensing architecture have many advantages: they are inexpensive, less susceptible to failure, robust against sensing uncertainty and useful for different applications. With the emergence of wireless sensor networks (Pottie and Kaiser 2000), a group of simple micro-robots also offers an attractive and scalable architecture for large-scale collaborative exploration of unknown environments.

Motivated by these considerations, we investigate several fundamental computational geometry problems in the context of simple robots, with minimal sensing ability. In particular, we assume that each robot is a (moving) point, equipped with a simple low-resolution camera that allows the robot to sense just the combinatorial features in its field. These features are unlabeled, however, meaning that the sensor can detect a vertex of the polygon but all vertices look the same to it.

The features are represented as two binary vectors, called the combinatorial visibility vector (cvv) and the point identification vector (piv), respectively. The cvv is defined by the cyclically ordered list of vertices visible from the robot's current position, where each bit indicates whether or not the consecutive vertices have a polygon edge between them. A 1-bit means that the two consecutive vertices are neighbors along a boundary component; a 0-bit means that the two consecutive vertices form a diagonal that hides a portion of the polygon from the current position of the robot. Figure 1 provides an ex-

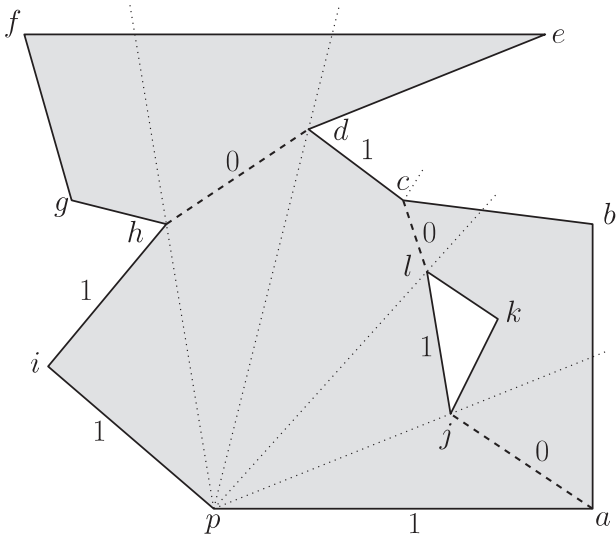


Fig. 1. Illustration of the combinatorial visibility vector. In cyclic (counterclockwise) order, the vertices visible from  $p$  are  $p, a, j, l, c, d, h, i$  and its visibility vector is  $cvv(p) = (1, 0, 1, 0, 1, 0, 1, 1)$ .

ample of a combinatorial visibility vector. A vertex might have peculiar visual features such as a robot standing on it, a pebble and so on. This information is encoded in the piv, which assumes the same ordering of vertices as the cvv. As already mentioned above, the robot senses only the combinatorial features of the polygon and has no access to geometric properties such as lengths or angles. For this reason we refer to this model as the *combinatorial sensing model*.

We assume that the visible vertices are seen in a consistent cyclic order. Without loss of generality, and to be concrete, we may assume that the order is counterclockwise around the robot's position. In particular, for the robot located on a boundary component at vertex  $v$ , the first vertex in the combinatorial visibility vector is  $v$ . For the robot located in the interior, we make no assumption about the identity of the first vertex; it is arbitrarily chosen by an adversary among the vertices visible from the robot's location.

It is worth pointing out, however, that the cyclic ordering is a local property, and does not assume an ability to distinguish between global clockwise and counterclockwise traversals of boundary components. Figure 2 illustrates this distinction. The cyclic visibility order at  $p$  is identical in both figures, yet following the boundary in the direction of the first edge leads to different global cyclic orientations (clockwise in Figure 2a and counterclockwise in Figure 2b). The robot has no other sensing capability and thus receives no information about distances, angles or world coordinates. In particular, for three consecutive points  $p, q, r$  along the robot's path, it cannot determine whether  $\langle p, q, r \rangle$  is a right turn or a left turn (namely, whether

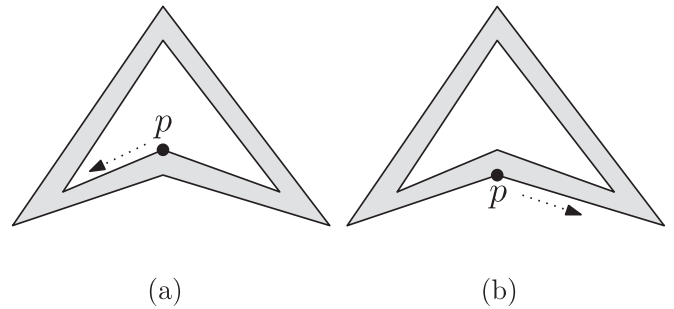


Fig. 2. The robots cannot distinguish a global cyclic orientation.

$r$  lies to the left or to the right of the directed line defined by  $p$  and  $q$ ).

We assume idealized (arbitrarily accurate) abstract sensing throughout, since our primary goal is to understand theoretical limits of our model. Our algorithms are deterministic and analyzed in the worst-case model.

We are interested in discovering what non-trivial or complex tasks such a robot can accomplish. Since the robot's information is local (the visibility region can be a small subset of the entire workspace), we investigate tasks that are seemingly global. For instance, can one or more robots decide whether a (unknown) polygonal environment (the robots' workspace) is simply connected, or it has holes? We show that a single robot with a single pebble – a marker that can be placed anywhere seen from distance and picked up again – is able to decide if the workspace is simply connected. Generalizing this, we show that if the workspace is a polygonal environment with  $k$  holes, then a single robot with one pebble can detect all the components of the polygon boundary.

Interestingly, in our minimalistic model of sensing, there are simple topological questions that are undecidable. For instance, while it is quite easy for a single robot to decide whether the workspace is a convex polygon, it is not possible to decide whether a given vertex is convex or reflex in a non-convex polygon. Similarly, a robot cannot decide which is the outer boundary component of the multiply connected polygonal workspace, although it can discover and count all the boundary components in the environment.

Finally, we show that a robot can compute a proper triangulation of a polygon using such local and minimalistic sensing. Furthermore, our triangulation algorithm can be implemented distributively by a group of  $m$  robots, each with  $\Theta(n/m)$  words of memory assuming that each word has  $\Theta(\log n)$  bits. As a corollary of the triangulation algorithm, we derive a distributed analog of the well-known Art Gallery Theorem (Chvátal 1975): a group of  $\lfloor n/3 \rfloor$  robots can self-deploy to guard a simple polygon. This improves the recent result of Ganguli et al. (2006), where they showed that  $\lfloor n/2 \rfloor$  distributed guards can achieve the coverage, raising the tantalizing question whether

the distributed nature of the problem is the source of the gap between the centralized optimum of  $\lfloor n/3 \rfloor$  and their bound. Our result shows that even with minimalistic sensing and distributed robots,  $\lfloor n/3 \rfloor$  guards suffice. Indeed, a triangulation is a fundamental data structure, used as a building block of many geometric algorithms, and so we expect that this basic result will find other applications as well.

### 1.1. Related Work

Combinatorial geometric reasoning is key to many motion planning and exploration tasks in robotics (Latombe 1991; LaValle 2006). Our work is similar in spirit to that of Tovar et al. (2007) and Yershova et al. (2005), in that we aim to explore the power and limitations of a minimal model of robot systems. Our model is different from that of Tovar et al. (2007) as they assume that important features of the environment are uniquely labeled, allowing sensors to distinguish these landmarks. Our model is similar to that in Yershova et al. (2005) but the nature of problems investigated in our paper is quite different from previously studied (Guibas et al. 1999; Sachs et al. 2004; Yershova et al. 2005). The main focus previously has been navigation and pursuit evasion. We are concerned, however, with the geometric and topological structure of the environment and collaborative and distributed self-deployment, which do not seem to have been addressed in the past.

## 2. The Combinatorial Sensing Model

### 2.1. Workspace

The workspace of the robots is modeled as a polygon. A polygon is a connected compact set  $P$  of the plane with the property that the topological boundary of  $P$  is composed of a disjoint union of non-intersecting piecewise-linear curves, called boundary components of  $P$ . If a polygon has a single boundary component, then it is called a simply connected polygon; otherwise it is called a multiply connected polygon (with holes). We refer to a simply connected polygon as *simple*. The polygon has an unknown geometry and an unknown number of vertices. The size of a polygon is the total number of its vertices.

### 2.2. Sensing Model

We consider robot systems with a simple model of ‘visual’ sensing. Whenever the robot is standing still at  $p$ , the sensory input of the robot is a combinatorial visibility vector  $\text{cvv}(p)$  and a point identification vector  $\text{piv}(p)$ . As long as the robot moves, it cannot sense. The  $\text{cvv}$  of a vertex is a cyclically ordered vector of zeroes and ones. This vector is defined by the vertices of the polygonal environment that are visible from the

point  $p$ , and the binary bits encode whether or not the consecutive vertices form an edge of the polygon (see Figure 1).

As mentioned earlier, we assume the visible vertices are always given in a consistent, say, counterclockwise, order around the robot’s position. Thus, the visibility vector tells the robot the number of vertices visible and a cyclic order of the edge types (boundary edge or diagonal) defined by consecutive vertices. Figure 1 shows an example, with the combinatorial visibility vector of vertex  $p$ . This consistent ordering enables us to speak about the right or left neighbor of a vertex  $v$ , which correspond to the first (not including  $p$ ) and to the last visible vertex from  $p$ , respectively. We abuse the notation by denoting a vertex  $u$  visible from  $v$  as  $u \in \text{cvv}(v)$ . Analogously, we say that  $u$  is the  $i$ th vertex of the  $\text{cvv}(v)$ , if  $u$  is the  $i$ th vertex in the cyclical ordering of the visible vertices from  $v$ .

The  $\text{piv}$  of a vertex  $p$  characterizes the potential extra information (e.g. the visual properties) on the vertices that are visible from  $p$  (including  $p$ ). The visible vertices are ordered as they are in the  $\text{cvv}$ . In our model, a robot can see whether a visible vertex is unoccupied or has another robot sitting on it or a pebble of a specific color, or a combination of these. We encode this accordingly with help of the set  $\mathcal{V}$  of visually distinguishable features. The  $i$ th component of the  $\text{piv}$  is a feasible (non-contradictory) subset of  $\mathcal{V}$ . Here we use  $\mathcal{V} := \{\text{empty\_vertex}, \text{rob}, \text{peb}_1, \text{peb}_2, \dots\}$ , where  $\text{empty\_vertex}$  denotes a vertex with no visual characterization,  $\text{rob}$  denotes a vertex with a robot on it and  $\text{peb}_i$  denotes a vertex with a pebble of color  $i$  on it.

Pebbles are marking devices that can be seen from afar; this is a standard tool used also in Yershova et al. (2005). Pebbles might have different colors that a robot can distinguish from a distance. A robot can place a pebble on the vertex it is currently on. Pebbles can be recollected for future use. Observe that since a robot on a vertex can be seen from afar, another robot can simulate the role of a pebble of a given color. Since we aim at a minimalistic model, our goal is to provide a robot with none or only a small number of pebbles (constant with respect to the size of the polygon) and of few distinct colors, say 1 or 2. Note that with this model, a robot cannot see another robot while the latter is moving.

We emphasize that the polygon induced by the visibility vector is different from the classical definition of the visibility polygon (Ghosh 2007); the former only uses the vertices of the original polygon, and is typically a strict subset of the visibility polygon. We believe that the visibility vector, despite being less informative than the corresponding visibility polygon, is better suited for our simple sensing model. In our coordinate-free combinatorial sensing model there is no obvious way to represent or communicate entities other than polygon vertices or edges. Even polygon vertices and edges have no ‘names’ or labels, and as such they can only be described in relative terms, e.g.  $i$ th vertex in the cyclic order of the visibility vector of vertex  $q$ . In this regard, our sensing model is more basic and weaker than the minimalism assumed by Tovar et al. (2007),

who require presence of labeled features and distinguishable landmarks in the environment.

### 2.3. Communication

In our distributed setting with multiple robots, we assume that robots are visually indistinguishable but possess distinct identifiers which can be learnt through communication. Different models of communication are conceivable, such as point communication, (limited) line-of-sight communication or global communication. The point communication allows two robots to communicate only if they are sitting at a common vertex. The (limited) line-of-sight communication enables the communication between two (close enough) visible robots. Finally, the global communication enables the communication between any two robots in the polygon.

Since we are interested in a minimalistic model, we adopt the point communication. In a communication, robots might send their IDs, motion orders (by specifying the index of the destination vertex in the  $cvv$ ), content of memory, etc. Observe that the uniqueness of the IDs enables the election of a leader among the robots sitting at a common vertex. Further, we assume that messages are sent without failures or interference.

### 2.4. Motion

In terms of the robot's motion abilities, we only assume that at a position  $p$ , a robot can choose to move on a straight line towards a vertex visible from position  $p$ . We assume that the robots always stop when the destination vertex  $v$  is reached and that it cannot maintain any information about the direction of the last motion that led to  $p$ . In particular, this implies that the robot cannot continue its straight motion along a line behind  $p$  (if a boundary component does not prevent this), along the so-called edge- (or diagonal- if the path to  $p$  is a diagonal) extension of  $p$  with respect to the origin of the motion.

Due to the consistent ordering of the vertices, the robot can move along a boundary component in a coherent direction by choosing either the right or left neighboring vertex as destination.

The general robot model is similar to that in Yershova et al. (2005) although that robot model enables them to walk on edge-extensions and does not involve collaboration. Gfeller et al. (2007) show that the ability to walk on edge-extensions enables the robots to solve problems that are impossible to solve otherwise.

In the following Lemmas we prove some simple facts about this combinatorial sensing model that shall allow us to use these operations as primitives for the robots in the rest of the paper.

**Lemma 1.** *A robot with a pebble can count the number of vertices of a simply connected polygon.*

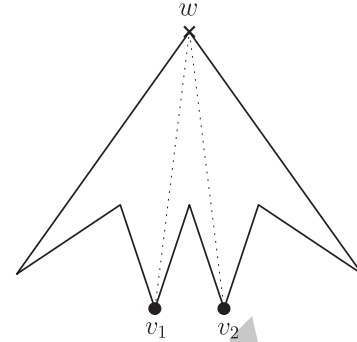


Fig. 3. A robot cannot recognize the origin of a movement from its destination.

**Proof.** The robot can easily count the number of vertices as follows. Start from an arbitrary vertex  $v$  and place the pebble on  $v$ . Move towards the right neighbor of the current vertex and increment a counter, until back at the pebble (i.e.  $piv_1 = \{\text{pebble}\}$ ). Since the boundary component of a simply connected polygon is connected, the number of motions of the robot equals the number of vertices and the algorithm terminates. ■

**Lemma 2.** *In general, if a robot without pebbles initially placed on the vertex  $v$  moves to vertex  $w$ , it cannot recover the position of the vertex  $v$  in the  $cvv$  of vertex  $w$ . It can recover this position if the robot has a pebble of a new color with respect to the pebbles already present in the polygon.*

**Proof.** Consider Figure 3 and suppose that a robot without pebbles can distinguish the origin of a movement, seen from the destination. Note that for the depicted polygon, we have that  $piv(v_1) = piv(v_2)$  and  $cvv(v_1) = cvv(v_2) = (1, 0, 0, 1)$ . Consider a robot without pebbles that moves from  $v_1$  to  $w$  in one instance and from  $v_2$  to  $w$  in a second instance. In both cases it starts with the same visual information at the origin and has the same visual information at the destination, since it does not have a pebble to influence the  $piv$  of  $w$ . It therefore concludes that the origin vertices are the same in both cases, which is false. On the other hand, if the robot has a pebble that uniquely marks the origin vertex, the task is easy. ■

### 2.5. Measure of Complexity

Movements are costly for robots in terms of energy. We therefore analyze the algorithms in terms of the total number of movements necessary to terminate an algorithm in a polygon with  $n$  vertices. The worst-case analysis is performed with respect to the polygonal shape, the starting position of the robots and their identifiers. We refer to this measure of complexity as the *running time* of an algorithm.

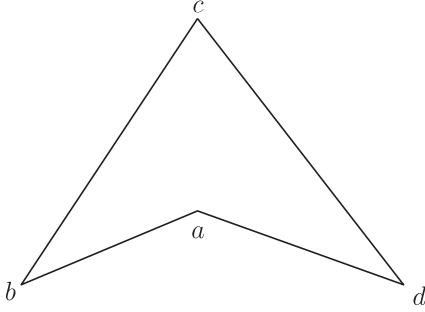


Fig. 4. An example showing that a robot in our model cannot resolve whether a vertex is convex or not.

However, the goal of this paper is to identify the problems that can be solved in the combinatorial sensing model and not to optimize the algorithm's efficiency. For this reason, the running times of the algorithms are stated only for the sake of completeness.

### 3. Convexity of the Environment

We begin with a simple example highlighting the severe limitations of our sensing model: a robot in our model cannot decide whether a given vertex of the environment is convex or reflex. Consider the symmetric polygon shown in Figure 4. The visibility vectors of the pairs of twin vertices  $\{a, c\}$  and  $\{b, d\}$  are identical:  $\text{cvv}(a) = \text{cvv}(c) = (1,1,1,1)$ , and  $\text{cvv}(b) = \text{cvv}(d) = (1,0,1)$ .

Suppose that an algorithm for deciding the type of an angle of a given vertex does exist, and apply it to the vertex  $c$ , specified in some way by the robot. Hence, the robot starts the algorithm execution from some vertex  $v \in \{a, b, c, d\}$  of the polygon depicted in Figure 4, and concludes correctly that  $c$  is convex. The decision is established by looking at the sequence of  $\text{cvv}$  and  $\text{piv}$  generated during the algorithm execution. By letting the robot start at the twin vertex of vertex  $v$  (i.e. at  $a$  if  $v = c$  or at  $b$  if  $v = d$ , or *vice versa*), it follows inductively that the sequence of  $\text{cvv}$  and  $\text{piv}$  generated is the same as before. The robot then specifies the vertex  $a$  and concludes that  $a$  is also convex, which is false.

Using only the visibility vector information available to our robot, it therefore cannot distinguish between the vertex types (convex or reflex) in our model.

This may seem surprising in light of the fact that such a robot can decide whether the entire polygon is convex or not. Observe that a necessary condition for the polygon to be convex is that the visibility vector of any vertex  $v$  must be all 1s—the presence of a 0 bit implies that there exists a ‘pocket’ in the polygon not seen by  $v$ . If the visibility vectors of all vertices are all 1s, then the polygon is convex. This follows because every non-convex polygon must contain a reflex vertex and if

$r$  is a reflex vertex then the visibility vector of either neighbor of  $r$  cannot be all 1s. An algorithm to determine the convexity of a polygon  $P$  follows.

Start at any vertex, say  $v_0$ , and consider the visibility vector  $\text{cvv}(v_0)$ . If this vector is not all 1s, stop: the polygon is ostensibly non-convex. Otherwise, compute the size of the polygon, say  $n$ , by counting the number of 1s in the vector. Repeat the convexity test from each of the remaining  $n - 1$  vertices, by moving cyclically along the boundary component. If and only if the visibility vectors of all the vertices are all 1s, the polygon is convex. We therefore have the following theorem.

**Theorem 1.** *A single robot in the combinatorial sensing model can decide whether a given polygon is convex or not in  $\mathcal{O}(n)$  steps. However, it is generally not possible to decide if a given vertex is convex in a non-convex polygon.*

## 4. Topological Structure of the Environment

We next consider another fundamental task related to the geometry of the environment: is the workspace of the robot simply connected, or does it have holes? If the polygon is multiply connected, then how many holes does it have? We first show that one robot, equipped with a single pebble, can decide the simplicity question. Subsequently, we show that with at least one pebble, the robot can also count the number of holes in the workspace and build a map that enables navigation in the polygon.

### 4.1. Is the polygon simply connected?

In order to decide the topological simplicity of its environment, the robot has to determine whether the 1-edges in its combinatorial visibility vector belong to two (or more) different boundary components. As an example, when located at vertex  $p$  (Figure 1), how does the robot decide whether the edge  $(l, j)$  is part of a ‘hole’ or does the outer boundary component connect to it in the back (near vertex  $k$  which is invisible to  $p$ )? We begin with a simple geometric fact that plays an important role in our algorithm. The *hole graph* of a polygon  $P$  is defined as follows. The boundary components of the polygon are the vertices of the graph and two vertices are connected by an edge, if and only if there are two mutually visible vertices of the polygon in the corresponding boundary components.

**Lemma 3.** *For any multiply connected polygon, the corresponding hole graph is connected.*

**Proof.** The proof follows from the fact that any polygonal domain admits a triangulation (using only polygonal edges and diagonals that connect vertices). Such a triangulation is a connected graph. By contracting each boundary component to a

single node and identifying all the diagonals between the same boundary components, the connectivity is preserved and we obtain the hole graph. ■

In particular, the previous Lemma states that for any boundary component  $C$  of a multiply connected polygon  $P$  there must be a vertex  $u \in C$  whose visibility vector includes a vertex from a different component of  $P$ . We utilize this fact to decide simplicity of the polygon as follows. The robot moves to a boundary component, and traverses it in cyclic order. At every vertex, it checks each vertex of its visibility vector to see if it lies on the same boundary component. By the preceding Lemma, if the polygon has a hole, then one of these tests will necessarily reveal this fact. The following pseudocode describes our algorithm for a robot with a pebble. The step that checks whether a vertex  $v$  is on the same boundary component as vertex  $u$  is done through a function EXPLORE, which is described right after the main algorithm.

#### SIMPLICITY ( $P$ )

1. Initially, move to an arbitrary vertex of a boundary component. Let  $u_1, u_2, \dots$  denote the vertices of this boundary component.
2. Count the number of vertices of the current component using the pebble. Let  $n_1$  be this number.
3. Begin the main phase of the algorithm. The vertex with the pebble currently on it represents the vertex being scanned. Let  $u_1$  be the initial vertex with the pebble. Repeat the following steps  $n_1$  times.
  - (a) Let  $u_i$  be the vertex with the pebble on it, and let  $\text{cvv}(u_i)$  be the visibility vector of  $u_i$ . Perform EXPLORE ( $u_i, v_j, n_1$ ) for each vertex  $v_j$  visible from  $u_i$ . (Move to vertex  $v_j$ , perform the EXPLORE operation, and then return to  $u_i$ .)
  - (b) If any call to EXPLORE returns  $\text{simple} = \text{false}$ , then terminate; the polygon is multiply connected. Otherwise, once all the vertices visible from  $u_i$  have been explored, advance the pebble from  $u_i$  to  $u_{i+1}$ .
4. After all  $n_1$  vertices have been explored and the algorithm has not terminated, report that the polygon is simply connected.

#### EXPLORE ( $u, v, n_1$ )

1. Starting at  $v$ , cyclically move counterclockwise along the boundary component containing  $v$ .
2. If a pebble is encountered within  $n_1$  steps, exit this call of EXPLORE (the current location is again the vertex  $u_i$ ).

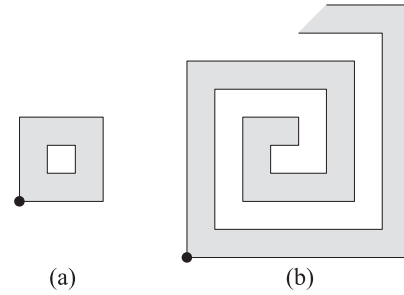


Fig. 5. A robot without pebbles cannot decide simplicity.

3. Otherwise, set  $\text{simple} = \text{false}$ ; retrace  $n_1$  steps backwards (returning to vertex  $v$ ) and move to the vertex  $u$  (identified with a pebble).

For example, in Figure 1 the call EXPLORE ( $p, a, 10$ ) does not discover a new boundary component, while EXPLORE ( $p, j, 10$ ) would do. We summarize this result in the following theorem.

**Theorem 2.** A robot with a pebble can decide if a polygon is simply connected or not in  $\mathcal{O}(n^3)$  steps.

The processing time of the algorithm can be improved in several ways, although that is not our main intent here. For example, instead of checking all vertices of the combinatorial visibility vector, it suffices to limit EXPLORE to only the first endpoint of the 0-edges.

The use of a pebble to mark a vertex in our algorithm seems critical. While we do not attempt to formalize this as an impossibility result, the following example (see Figure 5) suggests some of the difficulties. Suppose that an algorithm  $\mathcal{A}$  without using a pebble exists that correctly decides after  $t$  steps that the polygon in Figure 5a is non-simple. Running this algorithm from a vertex in the middle of the polygon depicted in Figure 5b (whose size is proportional to  $t$ ), the algorithm then has to conclude that this polygon is also non-simple, since at every step the two polygons yield identical visibility information. This argument is not completely satisfactory because the size of the polygon depends on the running time of the algorithm. However, it does suggest the difficulty in designing such an algorithm.

We next show how a robot can discover all the components in its workspace environment.

#### 4.2. Counting the Number of Holes

#### 4.3. The General Algorithm

The algorithm is a recursive extension of the simplicity algorithm: when the robot discovers a new hole, it recursively

scans its boundary component to discover new holes. Pebbles are used to mark the holes that have been discovered already. The key difference from the simplicity algorithm is that the robot needs to maintain state information to go back in the recursion. The following pseudo-code describes the algorithm by using pebbles of only two colors. An interesting aspect of this solution is that we do not require to mark every boundary component with a pebble of a new color as one might expect.

#### COMPONENTS ( $P$ )

1. Start on the boundary of an arbitrary component and compute its size. Let  $n_1$  be the size of the current component.
2. For the component being scanned, maintain the index (in cyclic order, from the starting vertex) of the current vertex, marked with a pebble of color 1. Let  $u_i$  be the current vertex. For each vertex  $v_j \in \text{cvv}(u_i)$ , in cyclic order, invoke  $\text{EXPLORE}(u_i, v_j, n_1)$ .
3. If  $\text{EXPLORE}(u_i, v_j, n_1)$  does not find a new component, then the scan of the current component continues.
4. If  $\text{EXPLORE}(u_i, v_j, n_1)$  finds a new component:
  - (a) Save the index of  $v_j$  in the cvv of  $u_i$  and the index of  $u_i$  in the cvv of  $v_j$ . For the latter,  $u_i$  is the only vertex with a pebble of color 1.
  - (b) Move back to  $u_i$ . Recollect the pebble of color 1 and replace it with a pebble of color 2.
  - (c) Use the pebble of color 1 to mark the new component and increment the component counter. Compute the size  $n'$  of the new component and set  $n_1 = \max\{n_1, n'\}$ , as the maximum length of the walk in  $\text{EXPLORE}$ .
  - (d) Recursively work on the new component. When exiting the recursion, move back to the location where the recursion was called. Replace the pebble of color 2 with the pebble of color 1.

By using Lemma 3, it is easy to see that this algorithm correctly discovers all the  $k + 1$  boundary components using a pebble of color 1 and  $k$  pebbles of color 2. The pebble of color 1 is required by Lemma 3 and marks the vertex currently being analyzed.

**Theorem 3.** *If the polygon consists of  $k$  holes, then a robot with a pebble of color 1 and  $k$  pebbles of color 2 can count the components in  $\mathcal{O}(n^3)$  steps.*

#### 4.4. Building a Map of the Polygon

The above-mentioned algorithm enables the robot to construct a data structure to describe the topology of the environment. The robot stores a spanning tree of the hole graph but with each vertex replaced with the boundary component and each edge attached to the proper boundary component vertex. By maintaining the navigation information, the robot builds the *navigation map* of the polygon, in which every vertex obtains a virtual label. This is achieved as follows.

The vertex where the algorithm has been started gets the label  $(1, 1)$  where the first component indicates the label of the boundary component. The vertices of the same boundary component are further cyclically labeled with  $(1, 2), \dots, (1, n_1)$ , where  $n_1$  is the number of vertices of the boundary component 1. Let  $(1, i)$  be the first vertex, for which  $\text{EXPLORE}((1, i), v_j, n_1)$  finds a new component. Then the robot maintains the information that the boundary component 2 can be reached from  $(1, i)$  by going to the  $j$ th vertex of its cvv. Accordingly, this vertex is labeled  $(2, 1)$ .

Furthermore, the robot recalls the index of  $(1, i)$  in the cvv of  $(2, 1)$  to be able to navigate back (by using the pebble of color 1). This proceeds naturally until all vertices of the polygon are labeled. Note that this can be performed concurrently to the COMPONENTS procedure. With this map (or the representation of the hole graph), a robot can move from every vertex  $u$  to every vertex  $v$  specified by their label.

#### 4.5. Discovering the Components with a Single Pebble

Using the part of this map already computed in the process of component finding, the number of pebbles can be reduced to 1 albeit at the expense of increasing the total amount of movement incurred by the robot in most cases. To see this, we modify the procedure  $\text{EXPLORE}$  as follows.

Suppose that the robot invokes  $\text{EXPLORE}(u_i, v_j, n_1)$ . Instead of leaving the pebble on  $u_i$  and resolving whether  $v_j$  belongs to the same component as the pebble, the robot puts the pebble on  $v_j$ . It then navigates through every vertex that it has already visited before (using the map) and verifies if the pebble is on one of them. If this is the case,  $v_j$  belongs to a known boundary component; otherwise a new boundary component has been found and the map can be updated. Note that this requires only the portion of the map that the robot has already built. This persistent backtracking removes the need to leave a pebble on any new boundary component.

**Theorem 4.** *A robot with a single pebble can count the boundary components of a (multiply connected) polygon and build the navigation map in  $\mathcal{O}(n^3)$  steps.*

In our algorithm, we assumed that the robot has  $\mathcal{O}(k)$  memory where  $k$  is the number of components required to save

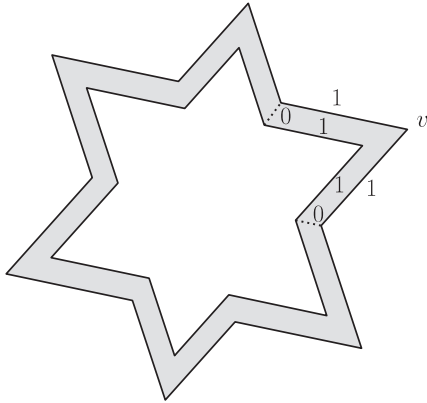


Fig. 6. An example showing that a robot in our model cannot resolve which is the outer boundary component.

the state of the recursion, which might correspond to the hole graph in the worst case. Alternatively, the same result can be obtained with  $\mathcal{O}(k)$  robots, each with constant (word) memory. Each robot stores a portion of the map while moving in coordination.

#### 4.6. Which is the Outer Boundary Component?

Finally, as another example of a simple task that is involved in our combinatorial sensing model, we argue that even although a robot can discover all the components (holes) in the environment, it cannot decide which one is the outer boundary component. Consider the centrally symmetric polygon shown in Figure 6. We observe that every vertex in this polygon has the same combinatorial visibility vector, namely  $(1, 0, 1, 1, 0, 1)$ , irrespective of whether it is on the outer cycle or the inner cycle. Because the robot cannot sense or measure angles or distances, its world view looks the same whether it is on the outer cycle or the inner cycle, implying that it cannot resolve between those two boundary components. Again, the proof exploits the fact that a robot executing an algorithm starting at a vertex of the inner boundary component generates the same sequence of cvv as a robot executing the same algorithm but starting at a vertex of the outer boundary component.

In Section 5, we show that a single robot using a constant number of pebbles is able to compute a proper triangulation of a simple polygon. This result, in addition to providing an important data structure for geometric reasoning and exploration, is also the basis for solving the art gallery problem in our sensing model.

## 5. Polygon Triangulation

We describe our algorithm for triangulating a simple polygon, which also is the key step in solving the art gallery problem

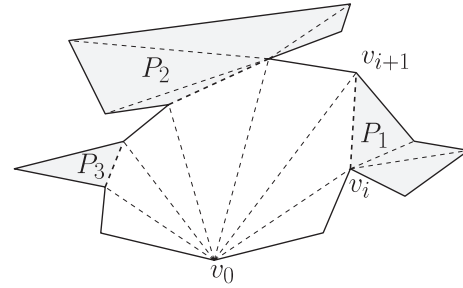


Fig. 7. The triangulation algorithm. Starting at  $v_0$ , the algorithm triangulates the pockets  $P_1, P_2, P_3$  in that order, and finally completes the triangulation by drawing edges from  $v_0$  to all vertices visible from it.

as a geometric structure with broad applicability (de Berg et al. 1997). We describe our algorithm for a single robot with a constant number of pebbles in the minimalistic model, with the assumption that this robot has  $\Theta(n)$  memory to store the triangulation and that it is aware that the polygon is simple (for instance, by running the algorithm *SIMPLICITY (P)*). Later we show that the same result can be obtained by a collaborative group of  $m$  robots, each with  $\Theta(n/m)$  memory. As a corollary of this distributed triangulation, we determine that a group of  $\lfloor n/3 \rfloor$  robots each with  $\Theta(1)$  memory can collectively build the triangulation and solve the art gallery problem.

Our polygon triangulation algorithm is recursive in nature, and works as follows (see Figure 7). The robot places an initial pebble at a vertex, say  $v_0$ . The combinatorial visibility vector of  $v_0$ ,  $\text{cvv}(v_0)$ , consists of a sequence of 0-edges intermixed with 1-edges. Each 0-edge is a diagonal that separates a ‘pocket’ from  $v_0$ , and the pockets defined by different 0-edges are pairwise disjoint. The robot will recursively triangulate the pockets formed by the 0-edges (say, by visiting them in cyclic order) and then complete the triangulation by drawing diagonals from  $v_0$  to all the vertices in its visibility vector.

In high-level pseudo-code, the triangulation algorithm therefore can be described as follows.

#### TRIANGULATION ( $P$ )

1. The robot begins at an arbitrary vertex  $v_0$ . Let  $e_1, e_2, \dots, e_k$  denote the 0-edges in the combinatorial visibility vector of  $v_0$  (in ccw order) and let  $P_i$  denote the pocket of the polygon defined by the edge  $e_i$ .
2. The robot recursively computes the triangulation of  $P_i$ , for  $i = 1, 2, \dots, k$ .
3. The robot finishes the triangulation by adding diagonals from  $v_0$  to all the vertices in its combinatorial visibility vector (endpoints of both 0- and 1-edges).

However, turning this high level description into a correct algorithm that fits in our combinatorial sensing model requires



several careful steps. We use the illustration of Figure 7 to explain the key steps of the algorithm.

At the top level of the recursion, the base visibility vector is defined for vertex  $v_0$ . The robot consistently scans the edges of a visibility vector in cyclic order. Let the first 0-edge in this visibility vector be  $v_i v_{i+1}$ , and let  $P_i$  denote the pocket (sub-polygon) defined by this edge. The vertices  $v_i$  and  $v_{i+1}$  are identified by the robot as the  $i$ th and the  $(i + 1)$ th vertices in  $\text{cvv}(v_0)$ . However, as the robot enters the pocket  $P_i$  for recursive triangulation, it no longer ‘sees’ the polygon from  $v_0$ , and needs a way to distinguish (identify)  $v_{i+1}$  from the ‘new base’  $v_i$  (see Lemma 2).

Similarly, the robot needs a way to recognize  $v_0$  from  $v_i$ , where it must return after the recursion ends in the pocket  $P_i$ . The robot does this using two pebbles as follows. (The location of  $v_0$  can also be stored by counting the number of boundary edges in the cyclic ordering. For the sake of uniformity, we use the pebbles to transfer this state information.)

**Lemma 4.** *Using two pebbles, the robot can compute two indices  $j_1$  and  $j_2$  such that  $v_{i+1}$  and  $v_0$  are, respectively, the  $j_1$ th and  $j_2$ th vertices in the cyclic ordering of the combinatorial visibility vector  $\text{cvv}(v_i)$ .*

**Proof.** From  $v_0$ , which is marked by the first pebble, the robot heads straight towards  $v_{i+1}$ , drops a second pebble there and returns to  $v_0$ . It then heads to  $v_i$  and identifies  $v_{i+1}$  as the first vertex in the cyclic order of  $\text{cvv}(v_i)$  that has a pebble; its index is the value  $j_1$ . Similarly, the index of the second vertex with a pebble serves as the value  $j_2$ . Having identified both  $v_0$  and  $v_{i+1}$  in the local visibility of  $v_i$ , the robot can then recover these pebbles, ensuring that we only use at most two pebbles throughout the algorithm. ■

Note that the procedure mentioned in the proof of the previous Lemma can be easily simulated by a robot with only one pebble, at the cost of an increased amount of total movement (as can be easily verified).

Once the robot can identify  $v_{i+1}$  from its local view, it knows the extent of the pocket  $P_i$ . The edge  $v_i v_{i+1}$  marks the end of the pocket and the recursive call to the triangulation.

Secondly, by always visiting the pockets in a cyclic order, the robot can consistently compute a ‘vertex labeling’ that serves to identify and store the triangulation globally. In particular, while at position  $v_0$ , the robot can see all the vertices in its visibility vector. This ‘view’ is entirely local; the  $j$ th ccw vertex in  $\text{cvv}(v_0)$  has no meaning to the robot when it is located at another vertex  $v_k$ . Thus, during the triangulation algorithm, the robot computes a global labeling. This is a cyclic ordering of the vertices in  $P$  starting from the base vertex  $v_0$ . This labeling is easily computed as follows.

The robot assigns the label 0 to the vertex  $v_0$  i.e.  $\ell(v_0) = 0$ . It then assigns increasing labels to all the vertices in  $\text{cvv}(v_0)$

until it comes to the first 0-edge, say  $e_i = (v_i, v_{i+1})$ . As the robot recursively computes the triangulation of  $P_i$ , it assigns labels in the pocket starting with  $\ell(v_i)$  and ending with the label  $\ell(v_{i+1})$ . At this point, the robot continues the labeling in  $\text{cvv}(v_0)$  until the next 0-edge, and so on. The triangulation is finally stored in the robot’s memory as a collection of diagonals. Diagonal  $(i, j)$  means the presence of a triangulation edge between the vertices that have indices  $i$  and  $j$  in the ccw walk along the polygon starting at  $v_0$ .

So far we have described the triangulation algorithm using a single robot with  $\Theta(n)$  memory (necessary to store the triangulation). However, it is easy to convert this into a distributed implementation using a group of  $m$  robots each with  $\Theta(n/m)$  memory. The single-robot algorithm can be simulated easily in this new setting: one robot acts as a leader and executes the triangulation algorithm while the others follow passively. They acting as storage devices and simulate the role of the pebble when required. This shows that  $\lfloor n/3 \rfloor$  robots, each with  $\mathcal{O}(1)$  memory, can achieve the triangulation.

**Theorem 5.** *In our combinatorial sensing model, one robot with a pebble can virtually label the vertices of a polygon and compute a triangulation of a simple polygon according to this labeling. The algorithm is easily turned into a distributed implementation using  $m$  robots without pebbles, each with  $\Theta(n/m)$  memory.*

In the previous algorithm, each robot moves for  $\mathcal{O}(n)$  steps which results in  $\mathcal{O}(n^2)$  total steps. There are instances of polygons where  $\Omega(n^2)$  steps are necessary to reach the guarding positions starting from a common vertex. A group of robots starting at one end of a winding polygon is such an example.

We have made no effort to ensure that the triangles stored by a robot are local in the sense that they are geometrically near to each other. However, a slight modification of the ordering in which the triangles are picked can easily ensure that every robot stores at most three triangles. When one of them shares a vertex with each of the other two,  $n = \mathcal{O}(m)$ . To do so, it is enough that the leader triangulates the pocket  $P_i$  it has just entered by picking ccw at every diagonal in  $P_i$ , starting from the last selected one during the last visit of  $P_i$  (or from the right neighbor, if  $P_i$  is visited for the first time) until the second endpoint of the next unvisited 0-edge. This is then visited recursively. The sequence of such triangles has the property that every triangle has a vertex in common with its predecessor and its successor.

To see this, we pick two successive triangles  $T_1, T_2$  of this sequence. If  $T_1$  and  $T_2$  are picked in the same recursion call, the claim follows trivially. If  $T_1$  and  $T_2$  are not picked in the same recursive call, then  $T_2$  must have been picked either (1) in a deeper recursion call than  $T_1$  or (2) after exiting the recursion call where  $T_1$  was picked. By construction, a triangle is immediately picked right after calling a recursive call, hence

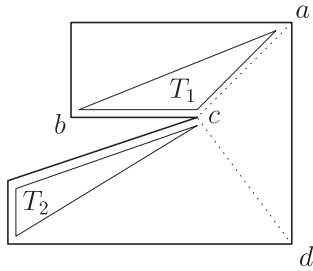


Fig. 8. This picture illustrates case (2) of the argument. The triangles have been drawn without using the polygonal edges and diagonals to ease the understanding.

$T_1$  and  $T_2$  share the first vertex of the 0-edge responsible for the recursive call for  $T_2$  and the claim is proved if (1) occurs.

The case (2) requires a more careful argument, because it is not guaranteed that the triangle  $T_2$  is picked immediately after exiting the recursive call of  $T_1$ . Suppose that  $T_1$  is specified by the vertices  $a, b, c$ , where  $a$  is the vertex where the robot is sitting on to triangulate the current region and  $b, c$  are two consecutive vertices of the cvv of  $a$ . We claim that the next triangle picked by the algorithm shares the vertex  $c$  with  $T_1$ . Let  $d$  be the vertex where the previous recursion was called. Thus  $a$  and  $c$  form a 0-edge in the cvv of  $d$ . If  $c$  is not the last vertex in the cvv of  $d$  in the current pocket, the claim follows trivially. Otherwise,  $c$  and  $d$  form a 0-edge of another node where the recursion was previously called and we can repeat the argument (see Figure 8). Since eventually  $T_2$  is going to be picked, this chain of exiting recursive calls stops and a triangle with the vertex  $c$  is picked.

It is therefore enough for the robots to store three contiguous triangles from this list to achieve the required property.

## 6. The Art Gallery Problem

The well-known Art Gallery Theorem (Chvátal 1975; Fisk 1978) asserts that every simple  $n$ -vertex polygon can be ‘guarded’ by placing  $\lfloor n/3 \rfloor$  guards at vertices of the polygon, and this bound is the best possible in the worst-case. The guarding positions are such that every point of the interior and the boundary of the polygon must be visible by at least one guard. The classical setting of the art gallery theorem assumes full knowledge of the polygon, including vertex coordinates. Ganguli et al. (2006) showed that given an unknown polygon,  $\lfloor n/2 \rfloor$  mobile guards (each with only local views) can ‘self-deploy’ at vertices to achieve the art gallery coverage. Their result raises the interesting question whether the gap between  $\lfloor n/2 \rfloor$  and  $\lfloor n/3 \rfloor$  is inherently due to the lack of global geometry. We show that this is not so, and in fact  $\lfloor n/3 \rfloor$  guards in our combinatorial sensing model can self-deploy to guard the polygon. Unlike in Ganguli et al. (2006), our algorithm is

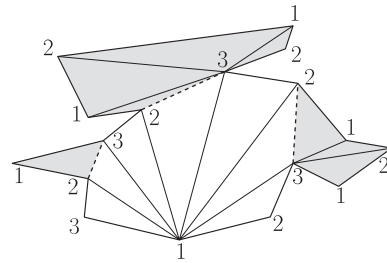


Fig. 9. Three-coloring the triangulation.

based on a triangulation mimicking the original proof of Fisk (1978).

The proof of Fisk (1978) uses the fact that a triangulation can be three-colored: three colors can be assigned to vertices so that no edge of a triangulation has the same color at both endpoints. Then, placing the guards at the least frequent color vertices solves the art gallery problem. There are  $n$  vertices, so the least frequent color occurs at most  $\lfloor n/3 \rfloor$  times. Since each triangle must have vertices of all three colors, every triangle is visible to some guard. Thus, to solve the art gallery problem in our sensing model, we just need to show that the triangulation computed in the previous section can be three-colored by our robots. Recall that the vertices have no coordinates, so we assign colors to the virtual labels of the vertices.

**Lemma 5.** *In the combinatorial sensing model, a robot can three-color the triangulation of a polygon by assigning colors to the virtual labels of the vertices.*

**Proof.** See Figure 9 for illustration. The robot begins by coloring the initial vertex  $v_0$ , from which the triangulation began, as 1. It then colors all the vertices in  $\text{cvv}(v_0)$  alternately as 2 and 3. Thus, each 0-edge in the visibility vector of  $v_0$  is colored (2,3). The robot now revisits the pockets  $P_i$  in the same order as in the triangulation step, and propagates the coloring. If the pocket  $P_i$  was triangulated by the robot from the vertex  $v_i$  and the color of  $v_i$  is 2, then the diagonals incident to  $v_i$  can be colored by alternating between colors 3 and 1, and so on. It is easy to see that this coloring succeeds, because both the diagonals picked by the robot and the polygonal edges receive different colors by construction. ■

Once colors have been assigned and recorded by the robot, it can count the vertices of each color and determine the least frequently used color. Placing guards at those  $\lfloor n/3 \rfloor$  vertices solves the art gallery problem. Since the robot labels the nodes sequentially (in cyclic order) around the boundary component, finding those nodes is trivial.

As before, we can implement this algorithm using a collaborative group of  $m$  robots, each with  $\Theta(n/m)$  memory. In

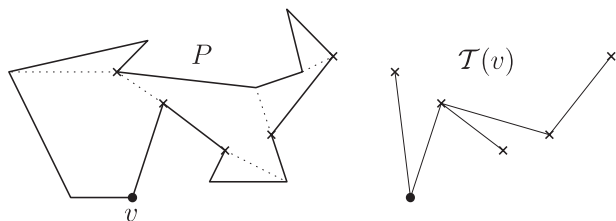


Fig. 10. The exploration tree of the polygon  $P$  with respect to  $v$ . The dashed diagonals are the 0-edges with respect to the vertices of  $\mathcal{T}$ .

fact, once the triangulation (along with the coloring that can be performed simultaneously) is in place (Theorem 5), the leader assigns every vertex of the least-used color to a robot which then can autonomously deploy.

**Theorem 6.** *In our combinatorial sensing model, a single robot with  $\Theta(n)$  memory and a pebble can solve the art gallery problem for an  $n$ -vertex polygon (determine the guarding positions). Alternatively, a group of  $\lfloor n/3 \rfloor$  robots each with  $\Theta(1)$  memory can solve the art gallery problem and self-deploy to guard the polygon.*

Since the main scope of this paper is to identify the solvable problems and not optimize efficiency to finding the solutions, the algorithm above has been deliberately chosen for its ease of description. Variants that improve on the total number of steps performed do exist. Recall that the focus of this paper is on the possibility of a robot to solve a problem rather than on the efficiency of doing so.

However, in the rest of the paper we describe an algorithm that performs better than the presented algorithm in many instances. The reason we do so is twofold. Studying the complexity of such problems might shed light on new structural aspects of polygons, so we take a first step in this direction. Additionally, we would like to present a result for which the collaboration among the robots plays a crucial role.

To better describe a variant of the algorithm, we introduce the concept of the *exploration tree*  $\mathcal{T} = \mathcal{T}(v)$  of a polygon with respect to a vertex  $v$ . This is a structure that allows a better understanding of the navigation in the above-presented algorithm. The exploration tree  $\mathcal{T}$  is recursively defined starting from the vertex  $v$ , which is the root of  $\mathcal{T}$ . Given a vertex  $w$  of  $\mathcal{T}$  consider the 0-edges of the cvv of  $w$  and order them ccw with respect to their endpoints. Add the first endpoint  $u$  of every 0-edge to the vertex-set  $V(\mathcal{T})$  of  $\mathcal{T}$  and the edge  $uw$  to the edge-set  $E(\mathcal{T})$  (see Figure 10 for an example).

$\mathcal{T}$  represents the robots' navigation scheme of the original algorithm (if we neglect the movements required by Lemma 4). Consequently, we can express the total number of steps by  $\mathcal{O}(n \cdot |E(\mathcal{T})|)$ . We now show how to reduce this

quantity to  $\mathcal{O}(n \cdot \text{depth}(\mathcal{T}))$ . First, recall that in this section we argued how a robot with a pebble can construct and navigate in  $\mathcal{T}$ . We therefore assume this capability without further comments to allow an easier presentation.

To design a better algorithm, observe that the coloring scheme presented above works without having the triangulation at hand. Every recursion pocket can be colored with only the information about the colors assigned to the endpoints of the diagonal defining the pocket: this is a local property.

To illustrate the idea of the improved algorithm, we consider a robot with  $\mathcal{O}(n)$  memory to explore the polygon and determine the guarding vertices. We show later how this can be simulated by  $\lfloor n/3 \rfloor$  robots with  $\Theta(1)$ -memory. Again, the robot uses the exploration tree  $\mathcal{T}$  as the navigation scheme: it performs a depth-first search in  $\mathcal{T}$  and executes the coloring algorithm discussed above. For every node  $w$  of  $\mathcal{T}$  it maintains the following state information:

- indices of the cvv to locate the parent of  $w$  in  $\mathcal{T}$  and the vertex of  $P$  that specifies the current pocket with  $w$ ;
- depth of  $w$  in  $\mathcal{T}$ ;
- total number of robots located on any node of  $\mathcal{T}^*$  (for the distributed version), where  $\mathcal{T}^*$  is the subtree of  $\mathcal{T}$  rooted at  $v$ ;
- triple  $\alpha(w) = (\alpha_1, \alpha_2, \alpha_3)$ , where  $\alpha_i$  represents the number of vertices of color  $i$  in  $\mathcal{T}^*$ ,  $i = 1, 2, 3$ .

Obviously  $\alpha(w)$  is obtained recursively:  $\alpha(w) = (\sum_{u \text{ child of } w} \alpha(u)) + (k_1, k_2, k_3)$ , where  $k_i$  is the number of vertices colored  $i$  in the current pocket (not including  $w$ ) and the sum is taken component-wise. Note that  $k_j = 0$ , for at least one color  $j$ .

As soon as  $\alpha(v)$  is known (recall that  $v$  is the root of  $\mathcal{T}$ ), the least frequently used color can be determined; suppose that this is color  $i$ . The recursive deployment of the robots can begin. Retrieve the  $i$ th  $\alpha$ -components of the children of  $v$  and send this number of robots in the corresponding pocket. The coloring algorithm is simulated again to recover the position of the vertices of the current pocket and some robots deploy to them. This procedure is called recursively for every further pocket. Obviously, the guarding positions established by this algorithm are the same as the original algorithm, hence correctness follows.

We show next how this algorithm can be simulated by  $\lfloor n/3 \rfloor$  robots with  $\Theta(1)$  memory each. The idea is to send only a minimal number of robots into a particular pocket, such that (1) every robot in the pocket will eventually guard a vertex in this pocket and (2) these robots maintain a chain between the vertex where the algorithm was executed (the 'robot's reservoir') and the robot deeper in a pocket. In case the number of robots inside a pocket is not sufficient for its complete exploration, this chain permits the efficient request of a new robot

to continue the task. We will make this precise in the following.

We subdivide the tree  $\mathcal{T}$  into trees of depth three. Start with the root of  $\mathcal{T}$  and let  $\mathcal{T}'$  be the subgraph induced by all the vertices of  $\mathcal{T}$  at distance at most three to the root. Remove  $\mathcal{T}'$  from  $\mathcal{T}$  (i.e. take the graph induced by  $V(\mathcal{T}) \setminus V(\mathcal{T}')$  which is the union of disjoint trees) and proceed recursively with all the remaining trees. We make every robot responsible for such a subtree  $\mathcal{T}'$  of  $\mathcal{T}$ , for which it maintains the state information of its root. Note that since the depth of  $\mathcal{T}'$  is constant, the robot can always maintain in its memory the information to reach the root of  $\mathcal{T}'$  from the vertex of  $\mathcal{T}'$  it is currently on. Hence, the state information of every node can be recovered from the state information of the leaves of  $\mathcal{T}'$ . Moreover, observe that  $\min_i |\alpha_i(u) - \alpha_i(v)| \geq 1$ , if  $u$  is a grandparent of  $v$ . So, a robot responsible for  $\mathcal{T}'$  can always find a guarding position within  $\mathcal{T}'$ .

We are ready to describe the algorithm. Initially, the robots are sitting at a common vertex  $v$  of  $P$ . A first robot is designed to be responsible for the maximal subtree  $\mathcal{T}'$  of depth 3 rooted at  $v$ . Every son of a leaf of  $\mathcal{T}'$  which is not a leaf of  $\mathcal{T}$  becomes a root of a new subtree and a new robot moves from  $v$  to it. The procedure then continues recursively. Any request for new robots, or any update for the information state that has to flow from a node of  $\mathcal{T}$  to the root, is transmitted via the robots responsible for the subtrees on the path to the global root. Since every branch is visited sequentially, this is always possible. Once the root of  $\mathcal{T}$  knows its state information, the final deployment of the robots to their guarding position starts. The set of robots waiting at a root are sent in the right amount into every branch of the corresponding subtree by the responsible robot.

To compute the total number of steps performed, note that every responsible robot has to walk along the edges of its subtree only twice (not including the steps required to deliver messages) and the union of each edge-set is  $E(\mathcal{T})$ . Further, every time a message has to be delivered  $\mathcal{O}(\text{depth}(\mathcal{T}))$  steps are sufficient. We crudely bound the number of messages required by the number of nodes of  $\mathcal{T}$ . Finally,  $n$  robots require  $\mathcal{O}(\text{depth}(\mathcal{T}))$  steps to reach the guarding position. Hence, the algorithm terminates within at most  $\mathcal{O}(n \cdot \text{depth}(\mathcal{T}))$  steps.

Unfortunately, the size of  $\mathcal{T}$  strongly depends in general on the choice of the first root, as Figure 11 shows. From the vertex  $u$  the whole polygon is visible, hence the exploration tree consists of only one point. On the other hand, the exploration tree is a long path if the root is taken to be  $v$  (the diagonals defining each pocket are represented by a dotted line).

## 7. Conclusions

We considered a simple and minimalistic model of visibility-based robot systems and showed that despite severe sensory limitations, these robots can accomplish fairly complex tasks

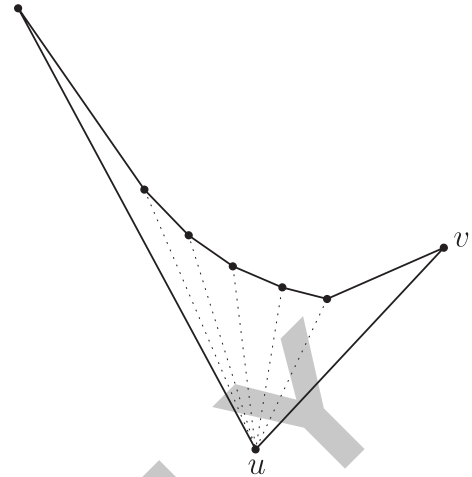


Fig. 11. The size of the exploration tree strongly depends on the choice of the root.

and infer global attributes of their workspace. At the same time, the impossibility of some seemingly simple topological tasks also highlights the limitations of our minimal model. It will be interesting to precisely identify the relationship between our model and the visibility graph as discussed in Ghosh (2007).

In future work, it will be interesting to explore which other global problems are solvable in this model and which ones are not. It will also be interesting to understand the inherent power of pebbles. We showed that a single pebble suffices even for discovering all the boundary components of a multiply connected polygon, but we do not know whether one can achieve these results without any pebble at all. Further, the motion capability assumed in Yershova et al. (2005) to walk along edge-extensions seems to increase the power of the robots significantly. An example of this can be found in Gfeller et al. (2007). A more exhaustive analysis of the capabilities of a robot with respect to the set of devices it is equipped with is described in Brunner et al. (2008).

It will also be useful to study the power of additional simple primitives such as local angle sensing, which circumvents the impossibility of deciding whether a vertex is convex and whether a component is the outer boundary component.

Finally, while our sensing model admits clean formulation and analysis, it obviously oversimplifies the limitations of real-world sensing. Real cameras have range limitations; they are unable to distinguish vertices that are nearly collinear, etc. It will be interesting to enhance our model to include some of these non-idealities.

## Acknowledgements

This article is the result of work done while the author was a visiting professor at the Institute of Theoretical Computer

Science, ETH, Zurich. The author wishes to acknowledge the support provided by the National Science Foundation under grants CNS-0626954 and CCF-0514738. The authors are partially supported by the National Competence Center in Research on Mobile Information and Communication Systems NCCR-MICS, a center supported by the Swiss National Science Foundation under grant number 5005 – 67322.

## References

- Brunner, J., Mihalak, M., Suri, S., Vicari, E., and Widmayer, P. (2008). Simple robots in polygonal environments: A hierarchy. In *Algorithmic Aspects of Wireless Sensor Networks, Fourth International Workshop, ALGOSENSORS 2008, Reykjavik, Iceland (To Appear)*.
- Chvátal, V. (1975). A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory*, **18**: 39–41.
- de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (1997). *Computational geometry: algorithms and applications*. Springer-Verlag.
- Fisk, S. (1978). A short proof of Chvatal’s watchman theorem. *Journal of Combinatorial Theory*, **24(B)**: 374.
- Ganguli, A., Cortes, J., and Bullo, F. (2006). Distributed deployment of asynchronous guards in art galleries. In *Proceedings of the American Control Conference*, pp. 1416–1421.
- Gfeller, B., Mihalak, M., Suri, S., Vicari, E., and Widmayer, P. (2007). Counting targets with mobile sensors in an unknown environment. In *Algorithmic Aspects of Wireless Sensor Networks, Third International Workshop, ALGOSENSORS 2007, Wroclaw, Poland, July 14, 2007, Revised Selected Papers*, Volume 4837 of *Lecture Notes in Computer Science*, pp. 32–45. Springer.
- Ghosh, S. (2007). *Visibility Algorithms in the Plane*. New York: Cambridge University Press.
- Guibas, L. J., Latombe, J.-C., LaValle, S. M., Lin, D., and Motwani, R. (1999). Visibility-based pursuit-evasion in a polygonal environment. *International Journal of Computational Geometry and Applications*, **9(5)**: 471–494.
- Latombe, J.-C. (1991). *Robot Motion Planning*. Norwell, MA: Kluwer Academic Publishers.
- LaValle, S. M. (2006). *Planning Algorithms*. Cambridge: Cambridge University Press.
- Pottie, G. J. and Kaiser, W. J. (2000). Wireless integrated network sensors. *Communications of the ACM*, **43(5)**: 51–58.
- Sachs, S., Rajko, S., and LaValle, S. M. (2004). Visibility-based pursuit-evasion in an unknown planar environment. *International Journal of Robotics Research*, **23(1)**: 3–26.
- Tovar, B., Freda, L., and LaValle, S. M. (2007). Using a robot to learn geometric information from permutations of landmarks. *Contemporary Mathematics*, **438**: 33–45.
- Yershova, A., Tovar, B., Christ, R., and LaValle, S. M. (2005). Bitbots: Simple robots solving complex tasks. In *Proceedings of AAAI National Conference on Artificial Intelligence*.