社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－AL－121 (7)
2008 / 12 / 3

A 4-competitive strategy for exploring unknown polygons

Xuehou TAN
Tokai University, 317 Nishino, Numazu 410-0395, Japan
tan@wing.ncc.u-tokai.ac.jp

Abstract. We present a new, on-line strategy for a mobile robot to explore an unknown simple polygon with n vertices, starting at a boundary point s, which outputs a so-called *watchman route* such that every interior point of P is visible from at least one point along the route. The length of the robot's route is guaranteed to be at most 4 times that of the shortest watchman route that could be computed off-line. This gives a significant improvement upon the previously known 26.5-competitive strategy, and also confirms a conjecture due to Hoffmann et al [5]. A novelty of our competitive strategy is a recursive procedure that reduces the polygon exploration problem to the subproblems of exploring two different types of reflex vertices, which are classified by their turning directions in the shortest path tree of s. The other is a mixture of techniques, including the off-line approximation algorithm for the watchman route problem [7], a geometric structure called the *angle hull* [5], and the paradiam for making the off-line techniques be on-line.

Keywords: Computational geometry; Competitive strategy; Watchman route problem; Polygon exploration problem; Angle hull

1 Introduction

In the last decade, visibility-based problems of guarding, surveying or searching have received much attention in the commumities of computatonal geometry, robotics and on-line algorithms. Finding stational positions of guarding a polygonal region P is the well-known *art gallery problem*. The *watchman route problem* asks for a shortest route along which a mobile robot can see the whole polygon P [6, 7, 8, 9]. If the shape of P is not known to the robot in advance, it introduces the *on-line watchman route problem* or the *polygon exploration problem* [1, 2, 4, 5].

When a starting point on the boundary of P is given, the shortest watchman route through s can be computed in $O(n^4)$ time using a dynamic programming algorithm [9]. An $O(n^5)$ time algorithm was later developed to remove the restriction of the given starting point s [6]. These two results have recently been improved to $O(n^3 \log n)$ and $O(n^4 \log n)$, respectively [3]. On the other hand, a simple, linear-time approximation algorithm for the watchman route problem with a given starting point s has been proposed [7], which reports a watchman route guaranteed to be at most $\sqrt{2}$ times longer than the shortest watchman route through s. For the watchman route problem without giving any starting point, the best approximation factor is 2 [8].

In the polygon exploration problem, a starting point s on the boundary of the polygon P is given. A robot with a vision system that continuously provides the visibility of its current position walks to see (or explore) the whole region of P, starting from s. Once a corner of P is seen, it is memorized forever, as the robot is able to draw the map of the polygon during its exploration. When each point of P has been seen at least once, the robot returns to s. We are interested in a *competitive* exploration strategy that guarantees that the route of the robot will never exceed in length a constant times the length of the shortest watchman route through s. For the problem of exploring unknown rectilinear polygons, a $\sqrt{2}$-competitive strategy has been presented [2]. For simple polygons, Deng et al. were the first to claim that a competitive strategy does exist, but the constant is estimated to be in the thousands [1]. A factor of 133 was later given by Hoffmann et al. [4], which has recently been improved to 26.5 [5]. Hoffmann et al. conjectured that the competitive factor is far below 10 [5].

In this paper, we give a new, on-line strategy for a mobile robot to explore an unknown simple polygon. First, we present a recursive procedure that effectively reduces the polygon exploration problem to the subproblems of exploring two different types of reflex vertices, which are classified by their turning directions in the shortest path tree of s. Our strategy for a subproblem is designed to follow the off-line approximation algorithm for the watchman route problem so that the robot's route is very close to the optimum watchman route [7]. To this end, a paradiam for making the off-line techniques be on-line is proposed. A geometric structure, called the *angle hull* [5], is also used in our analysis. With these new ideas, we are able to prove that an unknown polygon can be explored by a route of length at most 4 times that of the shortest watchman route through s. This gives a significant improvement upon the previously known 26.5-competitive strategy, and confirms a conjecture due to Hoffmann et al [5].

2 Preliminaries

Let P be a simple polygon and s a point on the boundary of P. A vertex is *reflex* if its internal angle is strictly larger than π; otherwise, it is *convex*. The *shortest path tree* of s consists of all shortest paths from s to the vertices of P. The vertices touching a shortest path from the right are called the *right reflex vertices*, or shortly, *right vertices*. The *left reflex vertices* or *left vertices* can be defined accordingly.

The polygon P can be partitioned into two pieces by a "cut" C that starts at a reflex vertex v and extends an edge incident to v until it first hits the polygon boundary. The piece of P containing s and including C itself is called the *essential piece* of C. We denote by $P(C)$ the essential piece of the cut C, and call v the *defining vertex* of C. See Fig. 1(a). A cut C_j *dominates* C_i if $P(C_j)$ contains $P(C_i)$ (Fig. 1(b)). We also say a point p *dominates* the cut C if p is not contained in $P(C)$. A cut is called the *essential cut* if it is not dominated by any other cuts. The watchman route problem is then reduced to that of finding the shortest route intersecting or visiting all essential cuts.

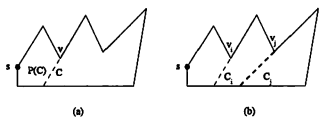

Figure 1: Essential cuts.

For ease of presentation, we denote by W_{opt} the shortest watchman route through s, and W_{app} the watchman route which is computed by the $\sqrt{2}$-approximation algorithm [7]. For a route R inside P, we denote by $|R|$ the length of R.

In the following, we briefly review the off-line $\sqrt{2}$-approximation algorithm [7], and then give the definition of angle hulls.

2.1 The off-line approximation algorithm

The reflection principle is used in most of the watchman route algorithms [3, 7, 9]. Let a and b denote two points on the same side of a line L. Then, the shortest path visiting a, L and b in this order, denoted by $S(a, L, b)$, follows the reflection principle. That is, the incoming angle of $S(a, L, b)$ with L is equal to the outgoing angle of $S(a, L, b)$ with L. The reflection point on L can be computed by reflecting b across L to get its image b', and then reporting the intersection point of L with $\overline{ab'}$. See Fig. 2(a). Let $L(a)$ denote the point of L closest to a. The path consisting of $\overline{aL(a)}$ and $\overline{L(a)b}$, denoted by $S'(a, L, b)$, gives a $\sqrt{2}$-approximation of the path $S(a, L, b)$, since the angle $\angle a\ L(a)\ b'$ is at least $\pi/2$ (Fig. 2(a)). The same result also holds for a line segment l. See Fig. 2(b).

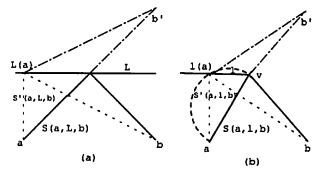

Figure 2: Approximating the reflection principle.

The idea of the $\sqrt{2}$-approximation algorithm is to repeatedly apply the approximation scheme designed for the reflection principle to essential cuts [7]. Let C_1, C_2, $\cdots$, C_m be the sequence of essential cuts indexed in clockwise order of their left endpoints, as viewed from s. Let $s = s_0 = s_{m+1}$. Given a point p in the polygon $P(C)$, we define the *image of* p *on the cut* C as the point of C that is closest to p inside $P(C)$.

Beginning with the starting point s, we first compute the images of s_0 on the cuts in the polygon P (or $P(C_0)$ [7]). Let s_1 denote the image of s_0 on C_1, s_2 the image of s_0 on C_2 and so on. The computation of s_0's images is terminated when the image s_{i+1} does not dominate the cuts $C_1, C_2, \ldots, C_i$ before it (Fig. 3). Then, we select a *critical* image from $s_1, s_2, \ldots, s_i$ as follows. If there exists an image s_h $(h < i)$ such that the image of s_h on C_{i+1}, which is computed in $P(C_h)$, dominates $C_{h+1}, \ldots, C_i$, we take the image s_h (e.g., the image s_1 in Fig. 3(a)) as the critical image. Otherwise, we take s_i (e.g., the image s_2 in Fig. 3(b)) as the critical image. Let s_k denote the chosen critical image. The images of s_k on the following cuts as well as the next critical image in the polygon $P(C_k)$ can similarly be computed [7]. This procedure is repeatedly performed until the image s_m on C_m is computed. See Fig. 3.

Let W_{app} denote the route which is the concatenation of the shortest paths between every pair of adjacent critical images (including s_0 and s_{m+1}). Clearly, W_{app} is a watchman route. A remarkable property of the route W_{app} is that the reflection points (i.e., critical images) of W_{app} on essential cuts are guaranteed to be to the left of those of the shortest watchman route W_{opt} through s. See also Fig. 3.

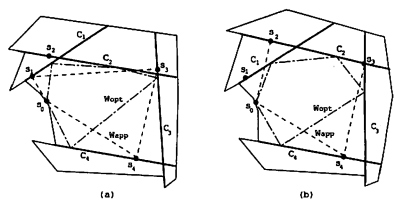

Figure 3: Critical images and routes W_{opt}, W_{app}.

Lemma 1 *(See [7]) Suppose that the image s_i on the cut C_i is critical. If the route W_{opt} reflects on C_i, then s_i is to the left of the reflection point of W_{opt} on C_i*

2.2 Angle hulls

In an unknown polygon, exploring a reflex vertex v requires a little care. Since we do not know the cut defined by v, the point on the cut closest to the current position of the robot, say, a, cannot simply be found. This difficulty is overcome by using the circle spanned by v and by a (see also Fig. 2(b)). Clearly, the intersection point of the circular arc with the cut is the point on the cut closest to a. This property leads to a study of angle hulls [5].

Let D denote a convex region in the plane. Suppose that a photographer follows a path to take a picture of D that shows as large a portion of D as possible but no white space or other objects, using a fixed angle lens, say, of 90°. All points enclosed by the photographer's path, and no other, can see two points of D at the right angle; we call this point set the *angle hull* of D, and denote it by AH(D). See Fig. 4(a).

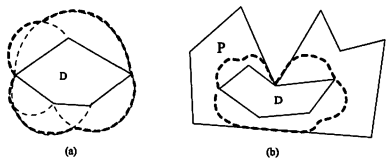

Figure 4: Angle hulls.

For the polygon exploration problem, the region D is defined as a *relative convex polygon* in P. That is, the shortest path between any two points of D inside P has to be contained in D. The photographer does not want any edges of P to appear in pictures; thus, the photographer's path may touch a vertex of P or overlap with a portion of the polygon edge. See Fig. 4(b).

In the outdoor setting, the perimeter of the angle hull is at most $\pi/2$ times the perimeter of D. In the indoor setting where D is contained in a simple polygon whose edges give rise to visibility constraints, we have the following result.

Lemma 2 *[5] Suppose that P is a simple polygon, and D is a relatively convex polygon (or chain) inside P. The length of the perimeter of AH(D), with respect to P, is less than 2 times the length of D's boundary.*

3 The 4-competitive strategy

In this section, we present a 4-competitive strategy for exploring an unknown polygon. It improves upon the strategy of Hoffmann et al. in the following two ways. First, we develop a 2-competitive strategy for exloring a subset of only right vertices or only left vertices. Our strategy resembles the off-line approximation algorithm [7], and thus, the robot roughly walks along the angle hull of the off-line approximation route. It then follows from the left-visiting property of the approximation route (Lemma 1) that the length of the robot's route is at most twice the length of the shortest watchman route that visits the cuts defined by that subset of reflex vertices. Second, we recursively reduce the polygon exploration problem to two groups of subproblems: one for exploring only right vertices and the other for exploring only left vertices. Our strategy selects a subproblem to solve as soon as it is possible. This gives the other factor 2.

For ease of presentation, we impose an ordering on the boundary points of P by a clockwise scan of the boundary, starting at s. When we say a boundary point u is "smaller" (resp. "larger") than the other point v, it implies that u is encountered before (resp. after) v by a clockwise walker on the polygon boundary, starting at s.

A vertex is said to be *discovered* if it has ever been visible once from the robot. A left or right vertex is *unexplored* as long as its cut has not been reached, and *fully explored* thereafter.

We call a polygon, the *right polygon*, if any clockwise tour that starts at s and always selects the smallest of the discovered right vertices to explore or visit gives a watchman route. Clearly, all essential cuts of a right polygon are defined by the right vertices, but the polygon with all essential cuts defined by the right vertices may not be a right polygon. (The concept of right polygons originates from a basic motion of our strategy that the robot makes a clockwise tour to explore as many as possible of the right vertices. See Section 3.2.)

In the following, we first propose a 2-competitive strategy for exploring a right polygon, and then describe in detail the 4-competitive strategy for exploring a simple polygon.

3.1 Exploring a right polygon

Let P_r denote a right polygon, and let s be a point on the boundary of P_r. An intuition of our exploration strategy is to explicitly compute all critical images in P_r. For this purpose, the robot makes a clockwise tour to explore all right vertices, in the order of them on the boundary of P_r. Clearly, whether a cut is essential can be determined when its defining vertex r is fully explored.

We denote by CP the current position of the robot, whose initial value is the point s. Denote by *RightTarget* the list of the right vertices ordered in clockwise order, which have already been discovered but not yet explored. Clearly, the list *RightTarget* has to dynamically be maintained, when the robot walks to explore a right vertex. Denote by r the head of *RightTarget*, which is the target vertex that the robot is going to explore. The value of r changes as soon as a smaller right vertex becomes visible from the robot.

In order for the robot to follow the off-line approxi-

mation algorithm [7], we further denote by CI the current critical image, whose initial value is also the point s. As in [7], the point CI will be considered as a local starting point in our strategy. Denote by C the *visibility* cut currently reached by the robot, and denote by TI the image of CI on the cut C. Since TI is the image of CI on C, we have $CI \neq TI$, except for the start setting where the very first right vertex is explored (see below).

A main difficulty arises in the polygon exploration problem is that we do not know the exact equation of the cut of the vertex being explored. More prepcisely, an on-line method for computing the set of critical images has to be given. For this purpose, we make use of the following two angle hulls, or circular arcs for short. Denote by $Cir(TI)$ the angle hull, defined in the region $P_r(C)$, of the line segment connecting r and the last vertex on the shortest path from TI to r. Recall that this shortest path is known to the robot, as a shortest path is always known between two points that have already been seen. Also, denote by $Cir(CI)$ the angle hull, defined in the polygon P_r, of the shortest path between r and the last vertex on the shortest path from CI to CP. When the robot (CP) moves, the last vertex on the shortest path from CI to CP dymanically changes, and thus, $Cir(CI)$ may dynamically be changed. As we will see below, the robot selects only the portion of $Cir(CI)$ in the region $P_r - P_r(C)$ to walk on.

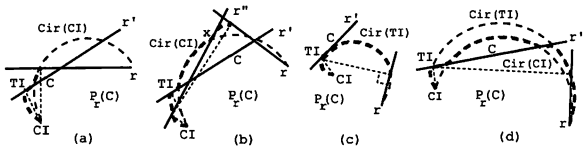

Figure 5: Basic motions in exploring the vertex r.

In the following, if no confusion is introduced, we simply refer to $Cir(TI)$ or $Cir(CI)$ as the clockwise oriented semicircle, which is spanned by r and by the last vertex described above. Let us consider how to explore the target vertex r. Suppose first that the right vertex r' has just been explored, and thus the cut of r' is taken as C. Assume that the target vertex r is visible from the robot, and the robot is now walking on the cut C to explore r, starting from the point TI. See Fig. 5. What should the robot do when $Cir(CI)$ or $Cir(TI)$ is encountered? If $Cir(CI)$ is met, the robot has to change its route to walk along the portion of $Cir(CI)$ in the region $P_r - P_r(C)$. This is because the image of CI on the cut of r may dominate the cut C. See Fig. 5(a) for an example, where the route of the robot is drawn in thick, dotted lines. The intersection point of $Cir(CI)$ with C has to be to the right of the (local starting) point TI, because r is larger than the defining vertex r' of C. On the other hand, while walking along $Cir(CI)$, the robot may encounter a cut that has been reached before (but it differs from C). In this case, the variable C is set to the encountered cut, the variable TI is maintained accordingly, and then the robot walks on the new cut C. This is because the old value of TI is known not to be a critical image [7]. For the example shown in Fig. 5(b), when the robot reaches the point x along $Cir(CI)$, the variable C is renewed to the cut of the vertex r'', the variable TI is renewed to the image of CI on the cut of r'', and then the robot walks on C (the cut of r''). It is also possible that the value of C is not changed, after the portion of $Cir(CI)$ in $P_r - P_r(C)$ has been traversed. See Fig. 5(d) for an example.

When $Cir(TI)$ is met during the walk on C, the robot changes its route to walk on $Cir(TI)$. This is because the point TI is a candidate for the next critical image. See Fig. 5(c). Since we have assumed that r is visible from the robot, $Cir(TI)$ is definitely encountered when the right endpoint (i.e., the defining vertex) of C is reached. Also, while walking along $Cir(TI)$, the robot may reach a cut (it may be C itself). In this case, the variable C is set to the encountered cut and the variable TI is maintained accordingly. See also Fig. 5(d) for an example.

The remaining case we have to deal with is that the robot reaches an intersection point i of C with a prior cut C', when it walks on C. In this case, if the point i is contained in the clockwise oriented semicircle spanned by r and by the last vertex on the shortest path from the image of CI on C' to r, the variable C is renewed to the cut C', and the variable TI is renewed to the image of CI on C'. This is because the new value of TI might be a critical image [7]. (If the point i is not contained in the semicircle described above, nothing is changed.) For the example shown in Fig. 6, when the robot walks along the cut of $r4$ to the intersection point i of the cuts of $r3$ and $r4$, the variable C is renewed to the cut of $r3$, and the variable TI is renewed to the point $s3$ (which is a critical image in the example shown in Fig. 6).

To complete the task of exploring the target vertex r, we assume that the variable TI is initially set to the point s, and that s is a special cut of P_r and its essential part $P_r(s)$ is the whole polygon P_r. This assumption helps explore the very first target vertex r, starting from s. That is, the robot repeatedly walks on the circle(s) $Cir(TI)$ to explore the first target vertex r. See the part of the robot's route from s to a in Fig. 6, which shows how to explore the very first target vertex $r9$.

Assume now that the target vertex r has fully been explored. What should we do next? Clearly, after the vertex r is fully explored, it is deleted from *RightTarget*. Note that on the way to explore r, the robot may cross the cut of the other right vertex; this vertex (which is larger than r and whose cut is dominated by CP) has to be removed from the list *RightTarget*, too. See Fig. 6 for an example, where the cut of $r8$ is crossed when the

robot moves from c and $s1$.

Whenever the cut of r (that has just been explored) is known to be essential, we determine whether a new critical image can be found. A critical image is found when the current point CP does not dominate all the cuts explored or visited by the part of the robot's route between CI and CP [7]. See Fig. 6 for an example, where the point $s1$ is recognized as a critical image when the robot reaches the point $s2$. As soon as a critical image is found, the variable CI as well as TI is maintained.

As our strategy explores as many as possible of the right vertices, it may happen that the robot loses sight of the next (discovered) right vertex, after the current target r is fully explored (i.e., r is the only right vertex visible from the robot at that time). In this case, the robot walks along the shortest path toward the head of *RightTarget* until it becomes visible again. For the example shown in Fig. 6, when the robot reaches the point $s2$, it loses sight of the target vertex $r4$. The robot then moves along the shortest path to the point e, so as to recover its view to $r4$.

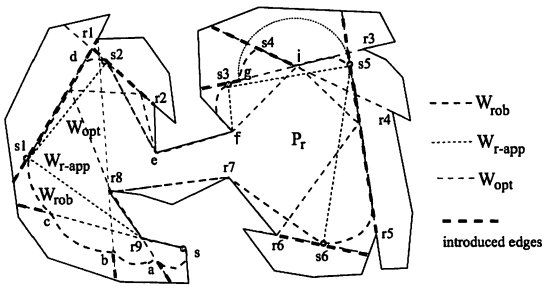

Figure 6: Exploring a right polygon.

The procedure for exploring the right polygon P_r, denoted by P_r-*Exploration*, is given below. The starting point s and the list *RightTarget* of right vertices, which are visible from s, are input of the procedure P_r-*Exploration*.

Procedure P_r-*Exploration* (**in** *RightTarget*, **in** s)

1. Set $CI, TI, C \leftarrow s$. Assume that s is a special cut and the essential part $P_r(s)$ is P_r.
2. **While** *RightTarget* is not empty **do**
   (a) The current *target* vertex is set to the head r of *RightTarget*. As soon as a smaller right vertex becomes visible, the value of r dynamically changes to the smaller one.
   (b) To explore the vertex r, the robot walks along (i) the portion of $Cir(CI)$ in $P_r - P_r(C)$ or $Cir(LI)$ (in $P_r(C)$) as soon as it is possible, (ii) the cut C if the robot is on C and visible from r, or (iii) the shortest path toward the vertex blocking the view of r.
   As soon as the last vertex on the shortest path from CI to CP (resp. from TI to r) changes, the circle $Cir(CI)$ (resp. $Cir(TI)$) is recomputed. When a cut is reached in the walk along $Cir(TI)$ or $Cir(CI)$, the variable C is maintained to the cut met by the robot, and the variable TI is also maintained. When an intersection point i of C with a prior cut C' is reached during the robot's walk on C, and when i is contained in the clockwise oriented semicircle spanned by r and by the last vertex on the shortest path from the image of CI on C' to r, the variable C is renewed to the cut C' and the variable TI is renewed to the image of CI on C'. Finally, *RightTarget* is maintained during the robot's walk.
   (c) When the cut of r (that has just been explored) is known to be essential, whether a new critical image can be found is computed. To be precise, a critical image is found when the current point CP does not dominate all the cuts explored along the route between CI and CP. After a new critical image is found, the variable CI (as well as TI) is maintained.
3. The robot returns to the starting point s along the shortest path.

Let us explain a little more on why Step 2 of P_r-*Exploration* works well. Step 2(a) specifies which vertex the robot is intending to explore. Step 2(b) gives the method to approach the cut of the target vertex and deal with the special case in which a cut intersection is encountered or the view of the robot to the target vertex gets blocked. Generally, the robot moves along the cut C to explore the target vertex r. As soon as the portion of $Cir(CI)$ in $P_r - P_r(C)$ or $Cir(TI)$ (in $P_r(C)$) is met, the robot changes its route to follow the encountered circle. On the other hand, when a cut is met during the robot's walk on $Cir(TI)$ or $Cir(CI)$, the variable C is maintained to the encountered cut, and the variable TI is maintained accordingly. Finally, Step(c) devotes to the computation of critical images. Fig. 6 gives an example to demonstrate how P_r-*Exploration* works (whose details are omitted here).

Lemma 3 *The set of critical images computed by the procedure P_r-Exploration is the same as that by the off-line approximation algorithm [7].*

Proof. Whether a cut of a right vertex is essential in the right polygon P_r can be determined when it is reached. In P_r-*Exploration*, the circle $Cir(CI)$ is used to compute the images of CI on the following cuts, and $Cir(TI)$ is used to determine whether TI is a critical image. Moreover, the variable C is dynamically maintained to the

cut, which is currently encountered by the robot and the image on it (i.e., TI) is a candidate for the next critical image. Hence, the lemma follows. □

Lemma 4 *Suppose that P_r is a right polygon with a point s on its boundary and RightTarget is the list of right vertices, which are visible from s. A call of P_r-Exploration(RightTarget, s) explores the polygon P_r by outputing a watchman route of length at most $2|W_{opt}|$.*

Proof. The main idea is to show that the robot roughly walks along the angle hull of the route W_{app} or W_{opt} (Lemma 1). See Fig. 6 for an example. Due to space limit, the detail of the proof is omitted in this extended abstract. □

3.2 Exploring a simple polygon

We present our competitive strategy for exploring a simple polygon P in a top-down manner. It mainly consists of two steps. At the first step, the robot makes a clockwise tour to explore as many as possible of the right vertices, with no attention to the exploration of the left vertices of P. The procedure P_r-*Exploration* can be used for this first tour, with a slight modification that the essential cuts and critical images are defined with respect to the set of explored right vertices. Clearly, all right vertices cannot be explored, as the left vertices of P are not explored.

The second and main step of our strategy is a recursive procedure for exploring the rest of the reflex vertices. Basically, the robot makes a counterclockwise tour to explore the left vertices. The left vertices, which were already discovered during the first tour of the robot, are initially put into the list *LeftTarget* in counterclockwise order. (Probably, some vertices of *LeftTarget* have been fully explored during the first tour.) Some of the right vertices having not yet been explored may become visible from the robot during this counterclockwise tour. Another idea of our strategy is to explore the newly discovered right vertices, as soon as it is possible.

Suppose that the current left vertex l is fully explored, and some right vertices have been discovered but not yet explored. (Note that some right vertices (e.g., $r5$ in Fig. 7) may fully be explored on the robot's way to the cut of l.) When should the robot switch to explore the right vertices? As the robot returns to its counterclockwise toure after a break for exploring right vertices, we define the *switching* condition to be that the non-essential part of the cut of l contains at least one newly discovered right vertex, but does not contain the head of *LeftTarget* (i.e., the next target left vertex).

Whenever the switching condition is satisfied, the symmetric procedure is called several times, so as to explore all the right vertices discovered up to now. It can be done as follows. First, find the shortest paths from s to the right vertices which have been discovered but not yet explored. Only those left vertices which are highest up in the shortest path tree of s are considered as the starting points for exploring the newly discovered right vertices. (Since the right vertices have been discovered, this portion of the shortest path tree of s is known to the robot.) Denote by *StartPoint* the list of those starting points. Next, the robot walks on the shortest paths to the points of *StartPoint* in counterclockwise order, and calls the symmetric procedure at each point of *StartPoint*. Note that during the robot's walk along these shortest paths, some left vertices (e.g., $l3$ in Fig. 7(b)) may be fully explored. After these procedures for exploring right vertices are terminated, the robot continues its counterclockwise tour to explore the remaining vertices of *LeftTarget*. In this way, the whole polygon P can eventually be explored.

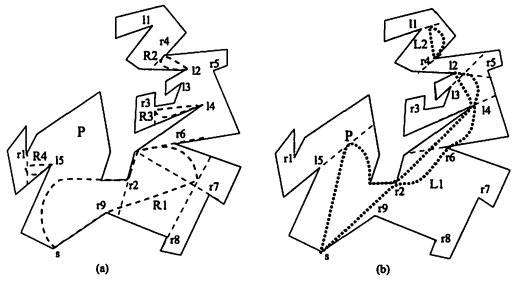

Figure 7: Exploring an unknown polygon.

Fig. 7 shows an example for exploring an unknown polygon. The routes $R1$ and $L1$ represent the first two routes for exploring the right vertices and the left vertices, respectively. At the time that the cut of $l2$ is reached (Fig. 7(b)), the robot switches to the route $R2$ so as to explore the right vertex $r4$ (Fig. 7(a)). Also, at the time that the cut of $r4$ is reached, the robot switches to the route $L2$ so as to explore the left vertex $l1$. Next, the robot moves to $l4$, switches to the route $R3$ so as to explore the right vertex $r3$, and so on.

Denote by P-*Exploration* the procedure for exploring a simple polygon, and *LeftExplorationRec* (resp. *RightExplorationRec*) the recursive procedure for exploring left (resp. right) vertices. To simplify the presentation, we omit the treatment for the left (resp. right) vertices that have been discovered but not yet explored during a clockwise (resp. counterclockwise) tour. Also, we assume that the procedure P_r-*Exploration*, which is used to complete the first clockwise tour, has been modified such that the essential cuts and critical images are defined with respect to the set of explored right vertices.

Procedure P-*Exploration* (**in** P, **in** s)

1. Initially, set *RightTarget* to the list of the right vertices, which are visible from s and ordered in clockwise order.

2. Call P_r-*Exploration*(*RightTarget*, s).

3. Sort in counterclockwise order all the left vertices, which are visible from the first clockwise tour of the robot, and then, put them into *LeftTarget*.

4. Call *LeftExplorationRec*(*LeftTarget*, s).

Let us further give the recursive procedure *LeftExplorationRec*. The point, at which *LeftExplorationRec* is invoked, is denoted by s_l. Note that the symmetric procedure *RightExplorationRec* is identical to *LeftExplorationRec*, except that left/right and clockwise/counterclockwise are exchanged. Also, the essential cuts and critical images computed by *LeftExplorationRec* or *RightExplorationRec* are defined with respect to the set of the left or right vertices explored in the same level, or by the same procedure *LeftExplorationRec* or *RightExplorationRec*.

Procedure *LeftExplorationRec* (**in** *LeftTarget*, **in** s_l)

1. Set $CI, TI, C \leftarrow s_l$. Assume that s_l is a special cut and the essential part $P(s_l)$ is P.

2. **While** *LeftTarget* is not empty **do**

   (a) The current *target* vertex is set to the head l of the list *LeftTarget*. As soon as a larger left vertex becomes visible, the value of l dynamically changes to the larger one.

   (b) To explore the vertex l, the robot walks along (i) the portion of $Cir(CI)$ in $P - P(C)$ or $Cir(TI)$ as soon as it is possible, (ii) the cut C if the robot is on C and visible from l, or (iii) the shortest path toward the vertex blocking the view of l.

   As soon as the last vertex on the shortest path from CI to CP (resp. from TI to l) changes, the circle $Cir(CI)$ (resp. $Cir(TI)$) is recomputed. When a cut is reached in the walk along $Cir(TI)$ or $Cir(CI)$, the variable C is maintained to the cut met by the robot, and the variable TI is also maintained. When an intersection point i of C with a succeeding cut C' is reached during the robot's walk on C, and when i is contained in the counterclockwise oriented semicircle spanned by r and by the last vertex on the shortest path from the image of CI on C' to l, the variable C is renewed to the cut C' and the variable TI is maintained to the image of CI on C'. Finally, *LeftTarget* is maintained during the robot's walk.

   (c) When the cut of l is known to be essential, whether a new critical image can be found is computed. To be precise, a critical image is found when the current point CP does not dominate all the cuts explored along the route between CI and the point CP. After a new critical image is found, the variable CI (as well as TI) is maintained.

   (d) Whenever the switching condition is satisfied, do the following:

      i. Set *StartPoint* to the list of the left vertices in counterclockwise order, which are highest up in the shortest path tree of s to all newly discovered right vertices.

      ii. **for** each vertex s_r of *StartPoint* **do**

         A. walk along the shortest path to s_r, and then set *RightTarget* to the list of right vertices in clockwise order, which are larger than s_r and have been discovered but not yet explored.

         B. Call *RightExplorationRec* (*RightTarget*, s_r).

3. The robot returns to s_l along the shortest path.

It is clear that neither left vertices nor right vertices can newly be discovered during the walks (Step A) in the **for** loops of *LeftExplorationRec* and *RightExplorationRec*. Except for the original point s, the local starting point s_l for *RightExplorationRec* (resp. s_r for *LeftExplorationRec*) is always a right (resp. left) vertex. Moreover, we can show that these starting points have to be visited at least once by the route W_{opt}.

Lemma 5 *All local starting points at which LeftExplorationRec and RightExplorationRec are called have to be visited at least once by the shortest watchman route W_{opt} through s.*

Prof. By symmetry, we only show that any local starting point s_r for calling *RightExplorationRec* has to be visited by W_{opt}. Recall that the left vertex s_r is highest up in the shortest paths from s to the right vertices, which are newly discovered by the previous call of *LeftExplorationRec*. So the cut of s_r has to be visited once by W_{opt}; otherwise, the cuts of the right vertices r, which are explored by calling the procedure *RightExplorationRec* (*RightTarget*, s_r), cannot be visited by W_{opt}, a contradiction. To visit the cuts of these vertices r, the route W_{opt} has to pass through the left vertex s_r (see Fig. 7); otherwise, moving the point of W_{opt} toward s_r on the cut of s_r produces a shorter watchman route, a contradiction again. Hence, the lemma follows. □

Let us now give a method to bound the total length of the robot's route. Observe that the robot's paths reported by a single call of P_r-*Exploration*, *RightExplorationRec* or *LeftExplorationRec*, without considering further calls within it, always form a closed curve. See Fig. 7 for an example, where the routes $R1$ to

$R4$, $L1$ and $L2$ denote the curves output by a single call of P_r-*Exploration*, *RightExplorationRec* or *LeftExplorationRec*. Also, we say that a call of P_r-*Exploration*, *RightExplorationRec* or *LeftExplorationRec* outputs a local watchman route, which is restricted to visit the cuts explored along the curve output by itself.

In the following, denote by R the route output by a call of P_r-*Exploration*, *RightExplorationRec* or *LeftExplorationRec*. Also, denote by R_{opt} the shortest route that visits all the cuts and the local starting points (at which further recursive calls are made within the procedure that reports R), which are visited by R.

Lemma 6 *For any route R, $|R| \leq 2|R_{opt}|$ holds.*

Proof. The route output by a recursive procedure *RightExplorationRec* or *LeftExplorationRec* differs from that by P_r-*Exploration* only at the very first walk in its **for** loop. (Note that the walk in the **for** loop contributes to the last part of the route reported by *RightExplorationRec* or *LeftExplorationRec*.) Since this walk is along a cut C that has just been reached, and since the defining vertex of C has to be visited by both routes R and R_{opt}, the portion of C walked by the robot can be counted as a part of the angle hull of the route R_{opt}. It then follows from the proof of Lemma 5 that any route R cannot exceed in length the perimeter of the angle hull of R_{opt}. Hence, we have $|R| \leq 2|R_{opt}|$. □

Lemma 7 *Let R and R' denote the routes output by two calls of P_r-Exploration or RightExplorationRec (resp. LeftExplorationRec). Then, R_{opt} and R'_{opt} are mutually invisible, with a possible exception for their starting points.*

Proof. Observe that the route R or R' output by a call of P_r-*Exploration* or *RightExplorationRec* (resp. *LeftExplorationRec*) explores as many as possible of the right (resp. left) vertices, and that they are separated by at least a route that explores the left (resp. right) vertices. The lemma follows from the switching condition and the optimality of the routes R_{opt}, R'_{opt} (see Fig. 7). □

Lemma 8 *Let W_{opt} denote the shortest watchman route through s. Then, the total length of the routes output by all calls of P_r-Exploration and RightExplorationRec (resp. all calls of LeftExplorationRec), is at most $2|W_{opt}|$.*

Proof. The following two observations on the shortest watchman routes can easily be made. First, the route W_{opt} is longer than the shortest route that visits only the cuts defined by right (resp. left) vertices. Second, any two routes R_{opt} and R'_{opt}, which are mutually invisible, cannot exceed in length the shortest route that visits the union of the cuts and the local starting points, which are visited by R_{opt} and R'_{opt}. It then follows from Lemmas 6 to 8 that all the routes output by calling P_r-*Exploration* and *RightExplorationRec* (resp. calling *LeftExplorationRec*) cannot exceed in length the perimeter of the angle hull of W_{opt}. It completes the proof. □

Now, we obtain the main result of this paper.

Theorem 1 *For a polygon P and a starting point s on the boundary of P, a call of P-Exploration(P, s) explores the polygon P, which outputs a watchman route of length at most 4 times the length of the shortest watchman route W_{opt} through s.*

Proof. A call of *P-Exploration(P, s)* produces two groups of routes, which explore the right vertices and left vertices, respectively. It follows from Lemma 8 that the total length of these routes is at most $4|W_{opt}|$. □

References

[1] X. Deng, T. Kameda and C. Papadimitriou, How to learn an unknown environment, *Proc. FOCS'1991* 298-303.

[2] X. Deng, T. Kameda and C. Papadimitriou, How to learn an unknown environment I: The rectilinear case, *J. ACM* **45** (1998) 215-245.

[3] M. Dror, A. Efrat, A. Lubiw and J. S. B. Mitchell, Touring a sequence of simple polygons, *Proc. STOC'2003*, 473-482.

[4] F. Hoffmann, C. Icking, R. Klein and K. Kriegel, A competitive strategy for learning a polygon, *Proc. SoDA'1997*, 71-82.

[5] F. Hoffmann, C. Icking, R. Klein and K. Kriegel, The polygon exploration problem, *SIAM J. Comput.* **31(2)** (2001) 577-600.

[6] X.Tan, Fast computation of shortest watchman routes in simple polygons, *Inform. Process. Lett.* **77** (2001) 27-33.

[7] X.Tan, Approximation algorithms for the watchman and zookeeper's problems, *Discrete Applied Math.* **136** (2004) 363-376.

[8] X.Tan, A linear-time 2-approximation algorithm for the watchman route problem for simple polygons, *Theoretical Comput. Sci.* **384** (2007) 92-103.

[9] X.Tan, T.Hirata and Y.Inagaki, Corrigendum to an incremental algorithm for constructing shortest watchman routes, *Int. J. Comput. Geom. Appl.* **9** (1999) 319-323.