Theoretical Robot Exploration

I. Volkan Isler

February 28, 2001

Abstract

The Shortest Watchman Route Problem, namely *'Given a polygon find the shortest route in the polygon with the property that each point in the polygon is visible from at least one point along the route'*, was solved in 1991. In order to model real-life situations, for example as encountered by a scout that must efficiently explore an unknown terrain, one must consider the online version of the problem. Exploration has been one of the major areas of research in Robotics but theoretical results on tractability, efficiency, and correctness of exploration algorithms are relatively recent.

In this report, we will review the approaches to explore an unknown environment. We will first present some essential concepts and review the lemmata that have been used in all exploration algorithms. Then, we will describe the solution to the Watchman Route Problem, since this algorithm forms the basis of all successive exploration algorithms. The first solution for the unknown environments was for the rectilinear case. We also consider the situation faced by a robot with a finite line of sight, which also has to return to the starting point every so often to refuel. The existence of a competitive algorithm for a rectilinear environment with an arbitrary number of polygons can be constructively refuted. Finally, we present the algorithm for a simple polygon with no obstacles and we conclude by discussing the open problems and future areas of research.

1 Introduction

How many guards do we need to cover the interior of an n-wall art gallery? This question posed in 1973 by Victor Klee is known as the art gallery problem. In 1975 Chavatal established the Art Gallery Theorem: $\lfloor \frac{n}{3} \rfloor$ guards are occasionally necessary and always sufficient to cover a polygon of n vertices. A slightly different version of the art gallery problem, known as Minimal Guard Coverage is: *Given a particular polygon, what is the minimum number of guards necessary to cover the polygon?* Suprisingly this problem turns out to be NP-Complete. All these results can be found in [12].

On the other hand, many shortest path problems like traveling salesman or hamiltonian cycle are well known to be NP-Complete. However, the combined problem, *given a polygon find the shortest route in the polygon with the property that each point in the polygon is visible from at least one point along the route* is known as the Shortest Watchman Route problem and was solved in 1991 [6]. This solution assumed that at least one point on the path is known. The problem with this assumption is that this path can be arbitrarily longer than the shortest watchman route without any restrictions. Nevertheless, Carlsson et al. presented a solution to the general problem [4] and recently the solution was improved by Tan who presented a $O(n^5)$ algorithm using dynamic programming [14]. Unfortunately, finding a shortest watchman route for polygons with holes and simple polyhedra is known to be NP-Hard[5].

In all of the problems above, the environment is assumed to be known a priori. In order to model real-life situations, for example as encountered by a scout that must efficiently explore an unknown terrain, one must consider the online version of the problem. Exploration has been one of the major areas of research in Robotics but theoretical results on tractability, efficiency, and correctness of exploration algorithms are relatively recent. Although it is known that there is no randomized competitive strategy against oblivious adversaries for a polygon with an arbitrary number of polygonal obstacles[7], the problem has been attacked by many researchers. A competitive algorithm for rectilinear polygons with bounded number of obstacles has ben presented[7], and it has been shown that there is no competitive strategy for a polygon with arbitrary number of obstacles [1]. Competitive algorithms for polygons without obstacles has been found recently[9, 10, 11] but exploration in more general environments remains to be solved.

In this paper, we will review the approaches to explore an unknown environment. In section 2 we will present the solution to the Shortest Watchman Route Problem, presented in [6], since this algorithm forms the basis of all successive exploration algorithms. This will allow us to present some essential concepts and review the lemmata that have been used in all exploration algorithms. Then in section 3, we will present the preliminary concepts for the online problem. In section 4 we present the solution for the rectilinear case. We also consider the situation faced by a robot with a finite line of sight, which also has to return to the starting point every so often to refuel. In section 5, we will present the construction that disproves the existence of a competitive algorithm for a rectilinear environment with arbitrary number of polygons. In section 6, we will present the algorithm for a simple polygon with no obstacles. We conclude the paper with open problems and future research.

2 Shortest Watchman Routes in Simple Polygons

In this section we will present an overview of the Shortest Watchman Route algorithm presented in [6]. The importance of this algorithm is that almost all the algorithms found later as well as the exploration algorithms make use of the concepts introduced in either this paper or the preceding papers of Chin and Ntafos that led to it. We will begin with some important definitions.

2.1 Preliminary Definitions

The following concepts are shared among all the algorithms we shall go over in this paper:

Definition 2.1 *By a polygon P, we mean an n-sided simple polygon P and its interior. The polygon can be described as a sequence of vertices $v_0, v_1, \ldots, v_{n-1}$ or a sequence of edges $E_0, E_1, \ldots, E_{n-1}$ indexed in the order they appear in a clockwise scan of P starting from the initial point s. E_i is oriented from v_i to v_{i+1}.*

Definition 2.2 *Point p is said to be visible from point s if and only if the line segment $\overline{ps}$ does not intersect with the exterior of P.*

A crucial definition needed is the one of a cut, also known as an extension.

Definition 2.3 *Let v_i be a vertex in P. The cuts C_{i-1} and C_i in P are the longest straight line segments that contain E_{i-1} and E_i, respectively and do not intersect the exterior of P. Each cut has the same orientation as the edge associated with it and it is described as an ordered pair $C_i = < s_i, t_i >$ where s_i and t_i are the starting and ending points of C_i.*

Cut C_i separates the boundary of P into two disjoint polygons. The polygon starting from s_i and ending in t_i in a clockwise scan is called LC_i and the other one is called RC_i. LC_i is sometimes referred as the foreign polygon, FP, of cut C_i.

Definition 2.4 *Cut C_i is called a visibility cut if s is in LC_i.*

A visibility cut is also referred to as a necessary extension of E_i, since if E_i is not visible from the current position, it will be so as soon as we cross its associated extension. Sometimes there is no way to visit an extension without crossing another. This allows us to concentrate on only the essential extensions.

Definition 2.5 *Cut C_i dominates C_j if the foreign polygon of C_i is contained in the foreign polygon of C_j. A cut is called an essential cut if an only if it is not dominated by any other cut.*

Note that any watchman route must visit all the essential cuts, otherwise by definition, there will be some unseen edge. We illustrate the concepts introduced in this section in figure 1.

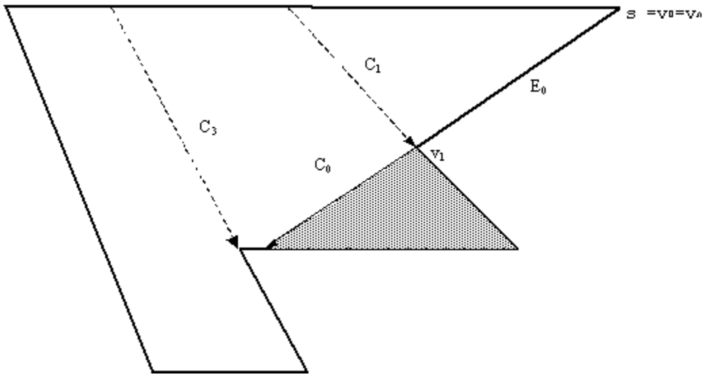

Figure 1: C_1 and C_3 are necessary extensions. C_3 dominates C_1, and it is non-dominated, therefore essential. Shaded area is the foreign polygon of C_0.

2.2 Shortest Watchman Route Algorithm

The key to find the shortest watchman route is to realize the fact that the polygon itself imposes an ordering on the essential cuts the route must visit. To understand this better we need the notion of a corner.

Definition 2.6 *A corner is an ordered set of essential cuts $C_i, C_{i+1}, ..., C_j$ such that:*

1. *each C_k intersects $C_{k-1}, i < k \leq j$,*
2. *C_i does not intersect C_{i-1}, and*
3. *C_j does not intersect C_{j+1}.*

An e-segment, also known as a fragment, is any line segment along the essential cut C_i that starts at s_i or at an intersection p_{ij}, ends at t_i or at an intersection p_{ik} and does not contain any intersections with other essential cuts in its interior.

In a corner with k essential cuts, there can be at most k-1 fragments along a cut, hence there can be at most $O(k^2)$ fragments in a corner. We need the notion of fragments because in general the notion of dominance between cuts may switch along the two sides of an intersection. The following lemma, which essentially states that a watchman route need not cross itself, is the essence of the constraint mentioned earlier.

Lemma 2.7 *There is a shortest watchman route that visits the set of corners in the order which they appear in a clockwise scan of the boundary of the polygon.*

The proof is by construction. Consider the crossing watchman route in figure 2. Whenever the route crosses itself, we can easily replace that part of the route with the one that visits the corners in order, without changing its length.

Lemma 2.7 helps us build a physical analogy for the process of finding a shortest watchman route. Imagine that the essential cuts are rods and there is a ring on each route that is free to slide along essential cuts. Now, consider a string that is threaded through the rings. Two ends of the string come together at the starting point s. Suppose that we pull the string taut and the threading is such that the string forms a convex chain in each corner. This is obviously a watchman route. Now, as we apply more and more force on the string, there will come a

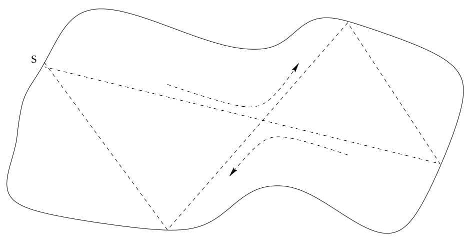

Figure 2: A noncrossing watchman route

time when no ring moves regardless of how much additional force is applied. At that point, the string traces a shortest watchman route. The algorithm presented in [6] simulates this physical process. The important issues in this simulation are not to become stuck at some local optimum and to find a short adjustment sequence.

The next step in the pursuit of a shortest watchman route requires us to take a look at how a watchman route can come in contact with the essential cuts.

Definition 2.8 *A watchman route can make a reflection contact with an essential cut if the route and the cut have exactly one point in common. A perfect reflection occurs when the angles formed by the incoming and outgoing sections of the route with the essential cut are equal. A crossing contact occurs when there are two points in common. Finally, they make a tangential contact if they share a line segment.*

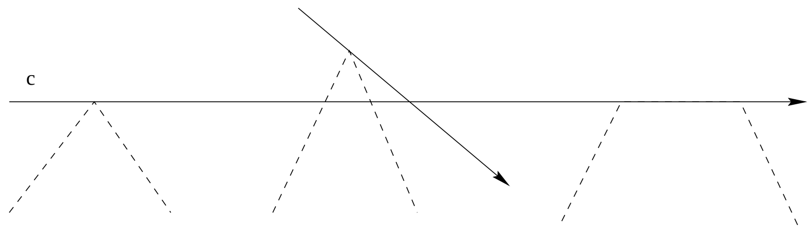

Figure 3: Three possible contacts of the watchman route(dashed line) with the essential cut c: (from left to right) a reflection contact, a crossing contact and a tangential contact.

The three different types of contacts are illustrated in figure 3. If the point of contact occurs at an internal point of the essential cut, then an optimum route must reflect perfectly on the cut. If the point of contact occurs at the intersection of two essential cuts, then the reflection need not be perfect with respect to either cut. In a crossing contact, the route passes through the essential cut C_1 and makes a reflection contact on some other cut C_2, and crosses C_1 again on its way out. The point of contact on C_2 must be to the left of C_1. Tangential contacts are degenerate cases of reflection contacts and occur when the portion of the route between two successive reflection contacts happens to overlap with an essential cut.

If a watchman route makes a reflection contact with a cut, the cut is referred to as an active cut and the fragment that contains the reflection is referred to as the active fragment. A shortest watchman route can be defined by the set of active fragments. If we had the "best" set of e-segments, we could find the watchman route using unfolding: The portions of P that are behind an essential cut(as viewed from s) are removed, resulting in a reduced polygon P'. Then P' is triangulated and rolled out by using the segment on the cut as a mirror and reflecting corresponding portions of the polygon with respect to these mirrors. The resulting polygon is known as the hourglass. Then the shortest path from s to its image in the hourglass gives the desired watchman route in P. This process is illustrated in figure 25 where the active segments are shown using dashed lines.

To recapitulate, the approach in [6] is to construct an initial watchman route R^0, then by checking local optimality properties at active fragments, to adjust the route so that

1. the route is made shorter after each adjustment and
2. a shortest watchman route is obtained when no more adjustments can be made.

We must now answer two questions:
I. How is the initial route obtained? For each corner we find a set of extended line segments so that, if a watchman route makes a reflection contact with these segments, it will visit all essential cuts in the corner. Algorithm *Navigation in a corner*, given below finds the extended line segments in a corner. Having found the extended line segments for all corners, the initial route R^0 is found by constructing a shortest path from s back to s that visits all the extended line segments using unfolding.
Navigation in a Corner: Initially, Let current cut be the first cut in some corner and s_i be the current initial point. Let Q be an empty queue. Repeat the following

```
Walk along C_i till an intersection is encountered.
If the next intersection is p_ij
        If there is an intersection p_ik to the left of C_i
                Walk along C_i
        Otherwise
                insert C_i to Q
                current cut = C_j
If the next intersection is t_i
        insert current cut to Q
        if current cut is the last cut
                break
        otherwise
                next cut in this corner becomes the current cut.
```

II. How are the adjustments made? When the incoming angle of R^m with C_j is smaller than its outgoing angle, it is possible to make R^m shorter by moving the point of contact to the left. This is illustrated in figure 4. Formally,

Definition 2.9 *Let $S_j^1, S_j^2, ..., S_j^{m_j}$ be the segments on the essential cut C_j and u_j^i be the common end point of S_j^{i-1} and S_j^i. A watchman route R^m is adjustable at u_j^{i-1} if and only if*

1. *R^m has a reflection contact with C_j at u_j^{i-1},*
2. *the incoming angle between R^m and C_j is smaller than the outgoing angle,*
3. *the route remains a watchman route if its contact with C_j is shifted to the left of u_j^{i-1}.*

There is also a symmetric case where the adjustment occurs at the other end and when the incoming angle is larger than the outgoing angle. Depending on the change of the number of active fragments, it is possible to have a -1,0,+1 adjustment. This is illustrated in figure 5. Note that if R^m is an adjustable watchman route then R^{m+1} is a watchman route.

We will need the following theorem to show that a sequence of adjustments leads to a shortest watchman route.

Theorem 2.10 *There is a unique non-adjustable watchman route in P.*

Theorem 2.10 is proved in [6] and it has two important corollaries.

Corollary 2.11 *A watchman route R is a shortest watchman route if and only if R is not adjustable.*

Corollary 2.12 *The shortest watchman route through s in a simple polygon is unique.*

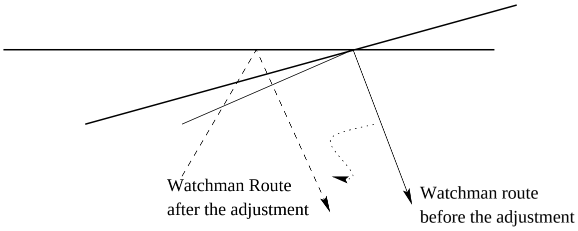

Figure 4: An adjustment can make a watchman route shorter

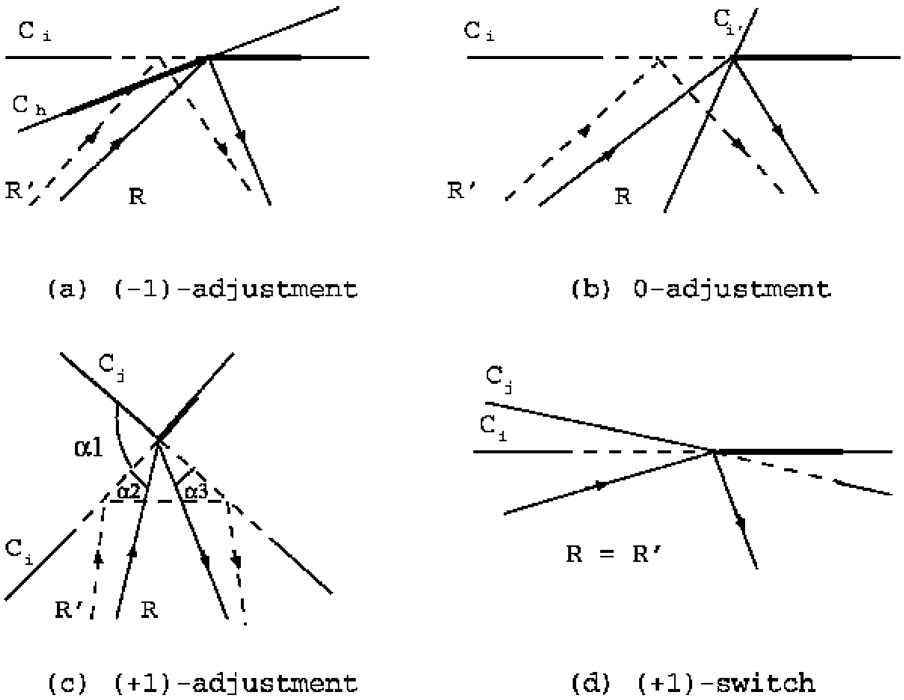

Figure 5: Different types of adjustments. taken from [14]

Now we have all the necessary results for the shortest watchman route algorithm: The algorithm proceeds by finding a point at which the current route is adjustable, updating the active fragments and constructing R^{m+1}. This is repeated until no further adjustments are possible. Since each adjustment results in a shorter route and the number of active segments is bounded, this process will eventually terminate. We conclude this section with the complete algorithm and its complexity analysis.

Algorithm Watchman Route

- Find all essential cuts and partition them into disjoint corners.
- Do *Navigation* in each corner.
- Obtain the initial route R^0 and the initial active segment set A^0. Let i,j = 0.
- While R^j us adjustable do
  - Pick the first segment that is adjustible in A^i and construct the next active segment set A^{i+1}
  - Use A^{i+1} to construct R^{i+1} as follows:

    * remove the portions of the polygon that lie outside the segments in A^{i+1}.
    * triangulate the resulting polygon
    * unroll the polygon using the extended segments containing the segments in A^{i+1} as mirrors
    * find a shortest path from s to its image s' in the unrolled polygon
    * fold the shortest path to obtain R^{i+1}.
  - Let j = j +1
- Report R^j as the shortest watchman route

Analysis: Navigation in corner can be performed in $O(n^2 logn)$ time and $O(n^2)$ space. Adjustments on previous cuts cannot oscillate along C_j. However, it is possible that the contact point with C_j will oscillate back and forth as adjustments on cuts with index higher than C_j are made. Since there can be at most $O(n)$ essential cuts with index higher than that of C_j and at most $O(n)$ segments along C_j, it follows that the total number of adjustments on C_j is at most $O(n^2)$. The total number of adjustments performed by the algorithm is at most $O(n^3)$. For each adjustment we need to construct the corresponding route which takes $O(n\, loglog\, n)$ steps. Thus the total complexity is $O(n^4\, loglog\, n)$ and this dominates the time complexities of other steps in the algorithm. Note that $O(n^2)$ space is needed in addition, to store the intersection points.

3 The online problem: Preliminaries

Now that we know how to construct the shortest watchman problem, we move on to the harder problem of exploration: Imagine that one day you open your eyes in flatland, in a room full of obstacles. Your task is to explore the room. Moreover you want to achieve this goal with minimum effort, you want to travel the shortest distance possible. To understand the task we are facing better, we have to answer two questions: What does it mean to explore a room and how do we measure efficiency?

In the previous section, we defined the a watchman route as *a route in the polygon with the property that each point in the polygon is visible from at least one point along the route.* It is easy to see that a path sees all parts of the interior perimeter of a simple polygon if and only if it sees all sides of the polygon, if and only if it sees all vertices of the polygon. Hence, any exploration algorithm must see all sides of a polygon.

Since the geometry of the environment is unknown, what we mean by an efficient algorithm may not be clear at first sight. We define efficiency in the style of online algorithms, algorithms that access the input data incrementally, and that must deal with events as they arrive without knowledge of future events. One way of measuring the performance of online algorithms is competitive analysis. In competitive analysis, we compare the performance of the online algorithm against the performance of the optimal offline algorithm and consider the worst case ratio. Let $cost_A(\sigma)$ be the cost incurred by an online algorithm A on the input sequence σ. Let OPT be the optimal offline algorithm and let $cost_{OPT}(\sigma)$ be the cost incurred by the optimal offline algorithm on input σ. We say that the online algorithm A is *c-competitive*, if there exits a constant b such that on every input sequence σ,

$$cost_A(\sigma) \leq c \cdot cost_{OPT}(\sigma) + b$$

The *competitive ratio* is the infimum over c such that A is c-competitive. More on online algorithms can be found in [8, 2]. Assuming that a starting point is given, in the absence of obstacles, we can compare the performance of our exploration algorithm with the optimum watchman route. If there are obstacles in the scene, or if the entry/exit points are possibly different unknown points, even the offline problem becomes NP-Hard. This puts the exploration problem in a different position than most online problems, where the corresponding offline problem can be solved efficiently. What we are looking for is an *online approximation algorithm for a hard problem.*

In watchman's route problem, since we have access to the whole polygon, we can exploit the structure imposed by the perimeter of the polygon: We know that there exits a shortest watchman route that visits the essential edges in order. In the online problem we don't have this luxury. In fact, although we can tell whether a cut is necessary or not when we first see it, we cannot tell if it is an essential cut until we explore its foreign polygon. Another difficulty in the nonrectilinear case is to estimate the direction of the cut of an unseen edge.

One way to attack a difficult problem is to restrict the input domain, to exploit the restrictions imposed on the input. In [7], Deng et al., provided the solution to the rectilinear case, where the edges of the polygon are aligned with one of the axes. We will present this solution in the next section.

4 Solution to the rectilinear case

A complete solution to the rectilinear case was presented by Deng et al. in [7]. The main result in this paper is a competitive strategy in the L_1 metric for a rectilinear, polygonal room with a bounded number of rectilinear, polygonal obstacles in it.

In the L_1 metric, the distance between two points (x_1, y_1) and (x_2, y_2) is given by $|x_2 - x_1| + |y_2 - y_1|$ as opposed to $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ in the L_2 metric. A useful fact is that an α-competitive algorithm in the L_1 metric implies a $\sqrt{\alpha}$-competitive algorithm in the L_2 metric. Note that the distance between (x_1, y_1) and (x_2, y_2) in L_1 metric gives the length of the rectilinear path from (x_1, y_1) to (x_2, y_2). The algorithm touches all the necessary extensions in order to see all sides of a polygon. In the rectilinear case, essential extensions have some special properties:

1. Two distinct essential extensions are either disjoint or perpendicular to each other.
2. Each essential extension intersects at most two other essential extensions. If it intersects two other essential extensions, these two are parallel to each other.
3. For any essential extension E, we can visit E, starting from the entry without visiting any other essential extension.

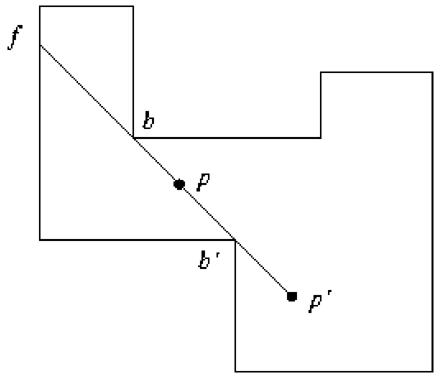

Figure 6: p can not see f, we say f^- has been seen. Taken from [7]

For the rectilinear case we use a slightly different definition of visibility. For two points a and b inside the polygon, if the line of sight $\overline{ab}$ intersects with the boundary, we no longer assume that a can see b. This is illustrated in figure 6. We use the notation f^- to specify the part of the edge E on which f lies that has been seen, where f is the intersection of E and the line ab.

Before we present the online algorithm for a rectilinear polygon without obstacles, we present two algorithms for the offline problem. The online algorithm will be obtained by imitating the second algorithm. The following naming scheme for the extensions and sides wil be used throughout this section:

The entry-exit point gives a natural order for extensions: $E_1, E_2, \ldots, E_m$ where E_1 is the first essential extension seen moving clockwise along the boundary from the start point. For each E_i we determine the unique side S and order them clockwise along the boundary as well: $S_1, S_2, \ldots, S_m$. If E is a necessary extension of two sides then we take the side that is closer to the entry. Note that E_i may not be an essential extension of S_i. However, we can permute E_i such that $E_{\sigma(i)}$ is an extension of S_i.

Next, we present the lemmata to be used in the design of the algorithms:

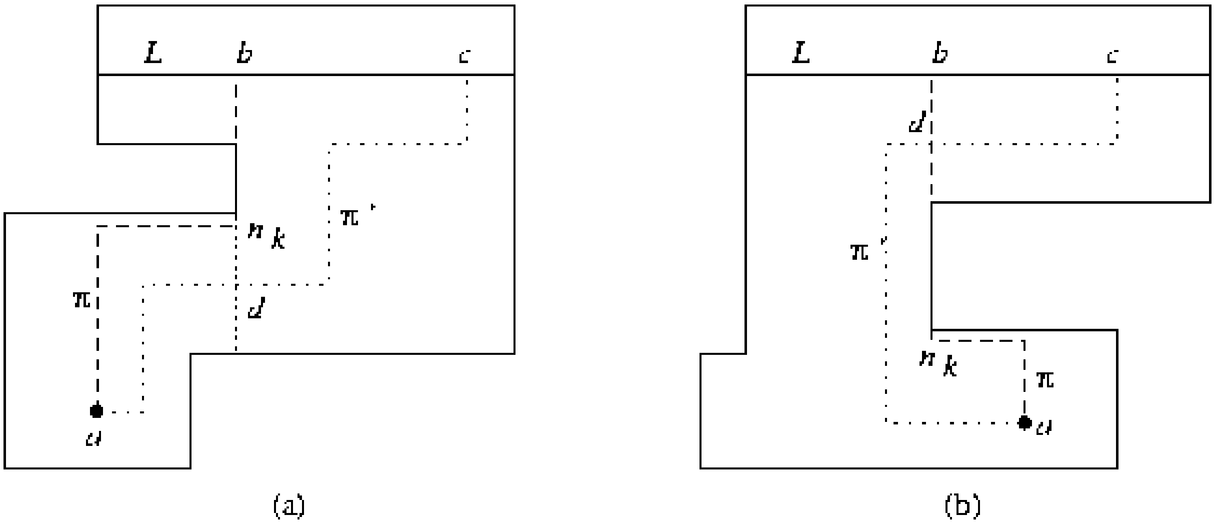

Figure 7: The shortest path π from a to L. Taken from [7]

Lemma 4.1 *Consider any line segment, L, parallel to one of the coordinate axes in a simple polygon P, which divides P into two subpolygons, and a be any point in P. Then all shortest paths from a to L meet L at the same point.*

Proof: If there is an obstacle free, straight path from a to L, then the lemma is trivially true. Otherwise, let π be a shortest path from a to L and b be the intersection of π and L. Note that the last piece of π must be vertical to L and its extension must overlap with a side of P (else, we could make it shorter). If, without loss of generality, we assume that L is east-west and a is to the south of L, the situation is as in either figure 7a or 7b. Now assume that there is another shortest path π', that meets L at $c \neq b$. Let S be the side contained in the extension of the last piece of π. In both of the cases we could make π' shorter by moving it closer to S, which leads to a contradiction.

Lemma 4.2 *Consider any line segment, L, parallel to one of the coordinates in a simple polygon P, which divides P into two subpolygons, and a be any point in P. Let π be a shortest rectilinear path from a to L. Then, for any other point $c \in L$, any rectilinear path from a to c is at least as long as $\pi \cup \overline{bc}$, where b is where π meets L.*

Proof sketch: By contradiction. Using the same assumptions as in the proof of 4.1, the situation is again represented in figure 7. Now assume that there is a path π' from b to c shorter than $\pi \cup \overline{bc}$. In both cases the contradiction follows from the observation that $|\pi'[d, c]| \geq |\pi[d, b] \cup \overline{bc}|$. Note that since π is a shortest path the portion if π before d must be greater than or equal to that portion of π'.

We are now ready to present the offline algorithms: Let x_0 be the entry point. For i=1,...,m let x_i be the point on E_i at the minimum distance from x_{i-1}. These points are uniquely defined by lemma 4.1 :

Algorithm Greedy Essential (GE_i) visits these points in the order $x_0 \to x_i \to x_{i+1} \to \ldots \to x_m \to x_1 \to x_2 \to \ldots \to x_{i-1}$. This is called a *Greedy clockwise path.*

Lemma 4.3 *$|GE_1| = |W^*|$ and $|GE_i| \le 2|W^*|$ where $|W^*|$ is the length of the optimal(shortest) watchman's route.*

Proof Sketch: Consider any W^* that visits E_i's in order. Let a be where W^* visits E_1. By lemma 4.1 we could replace $W^*[x_0, a]$ by $GE_1[x_0, x_1]$ without increasing the total length. The rest follows by induction. For GE_i $i \neq 1$, we consider W_1, the shortest path $x_0 \to E_i, E_{i+1}, \ldots, E_m$ and W_2, the shortest path $x_0 \to E_1, E_2, \ldots, E_{i-1}$. Also note that, $GE_i \le |W_1| + |W_2|$; $|W_1| \le |GE_1| \le |W^*|$ and $|W_2| \le |GE_1| \le |W^*|$.

If $\sigma(i) \neq i$ it is difficult to follow GE_1 online. Instead of visiting the essential extensions in order, algorithm greedy sides visits the extensions of the sides in order. Let $y_0 = x_0$ be the entry point. For $i = 1, \ldots, m$ let y_i be the closest point on $E_{\sigma(i)}$ from y_{i-1}.

Algorithm Greedy Sides(GS_i): visits the extensions of the sides in the order $S_i \to S_{i+1} \to \ldots \to S_m \to S_1 \to S_2 \to \ldots \to S_{i-1}$.

Lemma 4.4 *$|GS_1| = |W^*|$ and $|GE_i| \le 2|GS_1|$.*

Proof Sketch: To prove the above lemma, it is enough to show that the length of GS_1 is equal to $|GE_1|$. First we present the following observation about the structure of the path taken in greedy sides.

Proposition 4.5 *Let $\{x_i | i = 0, 1, \ldots, m\}$ and $\{y_i | i = 0, 1, \ldots, m\}$ be the points visited in greedy essential and greedy sides respectively. For each $i = 0, 1, \ldots, m$ one of the following holds:*

1. *$\sigma(i) = i$ and $|GE_1[x_0, x_i]| = |GS_1[y_0, y_i]|$.*
2. *$\sigma(i) = i + 1$, $\sigma(i + 1) = i$, $x_i \neq y_i$, $x_{i-1} = y_{i-1}$, $x_{i+1} = y_{i+1}$, E_i intersects E_{i+1} at y_{i+1} and $|GE_1[x_0, x_{i+1}]| = |GS_1[y_0, y_{i+1}]|$.*

Proposition 4.5 can be proven by induction on the number of the essential extensions. For the basis consider the situation in figure 8. Proposition 4.5 clearly holds. For the inductive step, we assume the proposition is true for $i = 1, \ldots, n$, then if we replace the entry by x_{i-1} in figure 8, we can show that the proposition holds for i+1.

Once we have proposition 4.5, we can show that the length of GS_1 is equal to $|GE_1|$ using induction and lemma 4.2. ∎

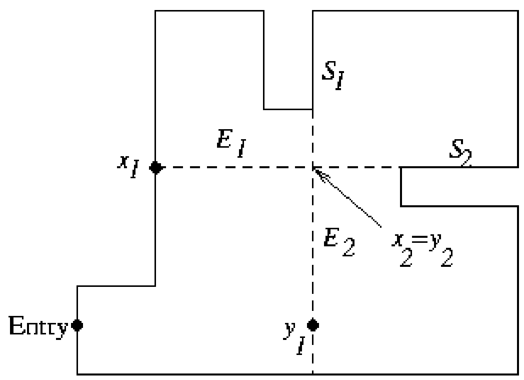

Figure 8: Greedy Sides vs. Greedy Essential. Taken from [7].

4.1 Algorithm for rectilinear polygons without obstacles

In this section we present the online algorithm in [7] for rectilinear environments with no obstacles. The algorithm tries to follow a set of rectilinearizations of the sections of the Taut-Thread path that starts at x_0, threads all the essential extensions and ends at x_0. Therefore it tries to imitate algorithm GS_1 presented earlier. First, some notation:

For point a on side S on polygon P, we define

- **F(a)**: the side of P that follows a.
- **B(a)**: the side of P that precedes a.
- **F'(a)**: the part of S after a.
- **B'(a)**: the part of S before a.
- **Ext(F(f))**: F(f)'s extension incident to f.

The algorithm uses the following variables:

- **M**: Current map of P, contains several disconnected pieces of P's boundary.
- **C**: Contiguous part of the boundary, starting at x_0, that has been seen so far.
- **Frontier f**: The end of C in clockwise direction.
- **z**: current position of the robot.

and the following auxiliary procedures:

- **ZZ2P(s,t)**: Finds a shortest rectilinear path from point s to point t.
- **ZZ2L(s,t,E)**: Finds a shortest rectilinear path from point s to a necessary extension E. t is the blocking, 270^o corner, and is implied by E.

Finally, we present the algorithm **Greedy Online:**
Initialization:
z ← z_0;
M ← parts of the boundary that are visible from z_0;
f ← initial frontier;
C ← part of P between z_0 and f.
Repeat the following steps updating M, C and f till M becomes a simple polygon. In that case follow the shortest path from z to z_0 and exit.

1. If f is at a 270^o corner then t ← f; E ← Ext(F(f));
   Otherwise b ← blocking corner when f^- was in view; t ← b; E ← Ext(B(b));
2. Form a taut-thread spanning z and t. Let $a_1, ..a_l$ be the points at which the threads touches corners of P. $a_0 \leftarrow z\ and\ a_{l+1} \leftarrow t$. If t is visible from z, l = 0.
3. if l ≥ 1, ZZ2P(a_j, a_{j+1}) $\forall j \in 0, \ldots, l-1$.
4. ZZ2L(a_l,t,E).
5. if t = b and f ∈ FP[Ext(B(b))] but not visible when Ext(B(b)) is reached, move along Ext(B(b)) till you see f or f^-.

A sample run of greedy online is illustrated in figure 9. The algorithm proceeds as follows. Initially the frontier is f_1. It is visible from z_0, so we have $l = 0$ and $ZZ2L(z_0, f_1, Ext(F(f_1)))$. As a result, the robot moves to Z_1. Now f_2^- is visible and b_1 is the blocking corner. We have $l = 0$ again, and we $ZZ2L(z_1, b_1, Ext(B(b_1)))$ as a result. Similarly the robot moves to z_3 and z_4. At the the fifth iteration, the robot is at z_4, current frontier is f_5 but f_5 is not visible. f_5^- was last visible from z_2, with b_2 being the blocking corner. We have $l = 0$ one more time, and $ZZ2L(z_4, b_2, Ext(B(b_1)))$ is invoked, updating the current position to z_5. The frontier jumps to f_6, and in step (2) we get $l = 3$, therefore the robot is moved to a_4 in step 2. Finally step 4 moves the robot to z_6. Now M becomes a simple polygon. In the last step (not shown) the robot moves from z_6 to z_0, completing the algorithm.

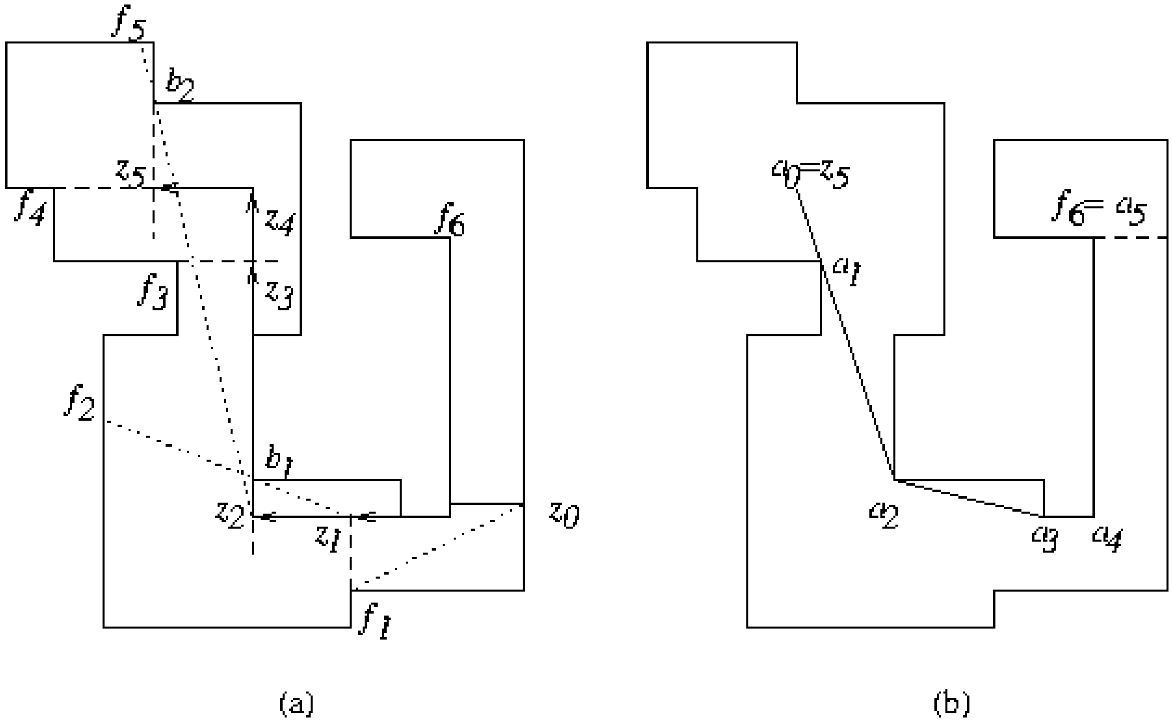

Figure 9: (a) ZZ2L invoked 5 times to reach z_5. (b) Determining a_l to visit f_6 from z_5. Taken from [7]

4.2 Analysis and optimality

The optimality of greedy online is obtained by showing that it has the same length as GS_1 (Recall that GS_1 is as long as the optimal watchman route). Note that greedy online has the following properties:

1. It terminates with C set to the boundary of the polygon P.
2. Let k be the number of times greedy online iterates. The extension N_i on which z_i lies is a necessary extension for $i \in [1, 2, \ldots, k]$.
3. The part of greedy online from z_i to z_{i+1}, $GO[z_i, z_{i+1}]$, is a shortest path from z_i to the necessary extension N_{i+1}.
4. $\{E_1, E_2, \ldots, E_m\} \subseteq \{N_1, N_2, \ldots, N_k\}$.
5. The essential extensions are reached in order $S_1, S_2, \ldots, S_m$

We will try to give some insight to why the above properties hold:

1. In step 1 of the algorithm either f or b is determined and in one iteration at least one edge is added to C. Since P has a finite number of sides, this process eventually terminates.
2. Every extension we visit is either Ext(F(f)) or Ext(B(b)). In the former case it is easy to see that F(f) has not been seen yet. The latter case is illustrated in figure 10. According to our claim $B(b) \in FP[Ext(B(b))]$, therefore the robot's current position p must be in the home polygon of B(b). Since f is the frontier there must be a wedge-shaped region (the dark region in figure 10), otherwise f^- would be invisible. Now, for contradiction, assume that p is in FP[B(b)]. Then the robot would have to enter this wedge-shaped region on its way from s, but this contradicts the fact that f is the frontier.

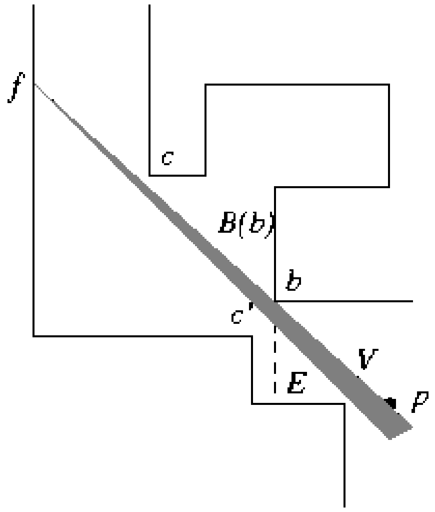

Figure 10: Illustration of the property 2 of Greedy Online. Taken from [7]

3. This is due to the fact that both ZZ2L and ZZ2P find the shortest paths.
4. First note that since it sees the whole polygon P, greedy online must visit every essential extension, by definition. Given this fact, we want to show that each essential extension is either Ext(F(f)) or Ext(B(b)). This is guaranteed by the fact that we will not cross or touch any other essential extension, other than N_i or N_{i+1} on our path from z_i to z_{i+1}. This is where orthogonality enters the scene, because all essential extensions are either perpendicular or parallel to each other. On our way, we cannot cross a parallel extension since in that case it would be dominated by N_{i+1}, therefore it can't be essential. On the other hand, we would not cross the perpendicular ones, because we take the shortest path from z_i to N_{i+1}.
5. This can be shown by induction on $\{S_i\}$ lying between the initial point and f. If we reach the next essential extension when t=f then this is indeed an essential extension satisfying property 5. Assume that Ext(B(b)) is the next essential extension. We claim that there are no essential extensions between f and b. Assume that there is an essential extension in between, parallel to b. This extension would dominate Ext(B(b)) therefore this is a contradiction. Next, Assume that there is an essential extension in between, perpendicular to b. Since t=b, this extension can not intersect Ext(B(b)), but then again it would dominate it. Hence, this also leads to a contradiction.

Once we have these properties, to show that greedy online is optimal, all that remains is to show that greedy online visits S_{i+1} via a shortest path from S_i. More formally we can state this requirement as the following:

Lemma 4.6 *Let $\{y_j | j = 0, 1, \ldots, m\}$ be the points defined for GS_1 and k be the total number of extension visited in greedy online. One of the following holds for each $i : i = 0, 1, 2, \ldots, k$.*

- *We reach N_i at z_i via a shortest path from the last essential extension visited; or*
- *We reach N_i at the intersection of N_i and the last necessary extension that is perpendicular to N_i via a shortest path from the last essential extension visited.*

It is possible to prove lemma 4.6 using induction on i. Although straightforward, the actual proof is laborious, therefore we refer the reader to the original paper [7].

We summarize the discussion above with the following theorem:

Theorem 4.7 *Greedy Online has the same length as greedy sides, that is $|GO| = |GS_1|$.*

4.3 Algorithm for rectilinear polygons with obstacles

In this section, we will take one step further and present an algorithm for a rectilinear environment with a bounded number of obstacles in it. Exploring the obstacles requires an algorithm for viewing the exterior of an object. However, viewing the exterior is related to viewing the interior in the following way: Consider the situation in figure 11. The polygon P' is obtained by constructing a large enough rectangle that contains P. Then we pick a random point on P and add a narrow passageway connecting the interior of P' and exterior of P. An optimal path that sees the interior of P' is at most twice as long as the one that sees the exterior of P.

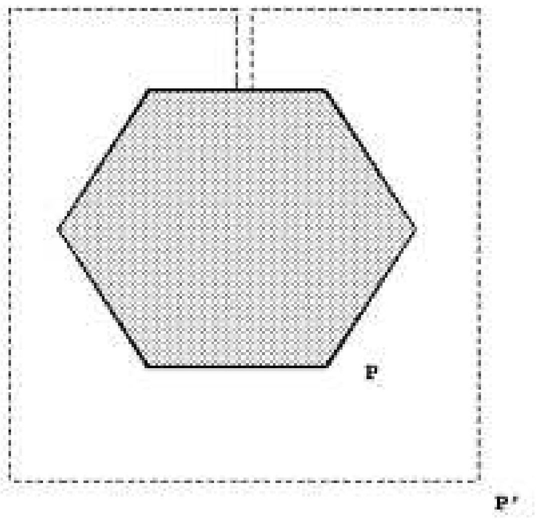

Figure 11: Viewing the exterior of a polygon: A path that sees the interior of P' sees the exterior of P.

In the rectilinear case, viewing the exterior has a special property which is stated in the following lemma.

Lemma 4.8 *There is an optimal watchman route for viewing the exterior of a rectilinear polygon such that the route encircles the polygon.*

Lemma 4.8 is illustrated in figure 12. Given a watchman route $W_1 \cup W_2$, we can choose a rectilinear path P such that $P \cup W_1$ encircles the polygon. In the rectilinear case the length of $W_1 \cup P$ is no longer than $W_1 \cup W_2$. This does not hold for the nonrectilinear case, in figure 12-b we can make the length of the encircling path arbitrarily large by changing the height of the triangle. In this case path W will still see the exterior of the triangle.

For external search, we define an *external extension* similar to the internal extensions. The only difference is the direction of the extension; we extend to the exterior of the polygon. Hence some of the extensions can be infinite. We define a *necessary external extension* in the following way: Consider a taut thread around the given polygon starting from and ending at x_0. We ignore all external extensions that are crossed by this thread. Note that it is not possible to have an infinite necessary extension. An *essential external extension* is a non-dominated external necessary extension, as expected.

External search differs from the internal search in a couple of ways. First of all, if there is no essential extension in the interior search, we are done since the whole polygon can be seen from x_0. On the other hand, even if there are no essential extensions, we still have to go around the polygon in the external search. As a result, in the external search, the robot may not have the complete map of the polygon even after visiting the last essential extension. In fact, it may not even know that it has reached the last necessary extension. Nevertheless, based on the essential extensions and the Taut-Thread principle, the exterior of a rectilinear polygon can be explored

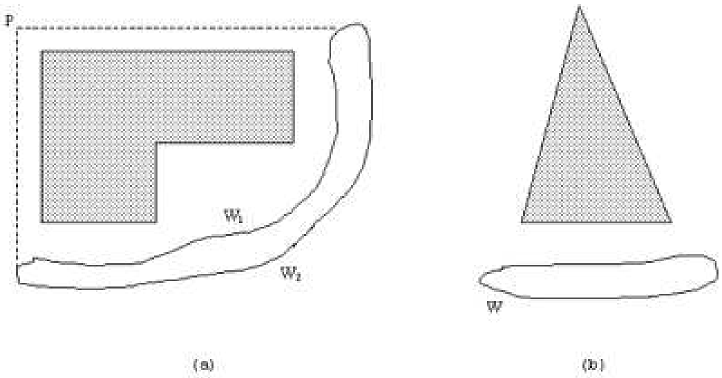

Figure 12: (a) In the rectilinear case, a watchman route that encircles P can be constructed from any watchman route: we break the watchman route into two parts W_1 and W_2 and create the path $P \cup W_1$. (b) This construction is not valid in the non rectilinear case(see text).

essentially the same way as its interior. To see this, we will need the following lemma, the proof of which carries over from Lemma 4.3:

Lemma 4.9 *Let $E_1, E_2, \ldots, E_t$ be the external essential extensions with respect to the entry-exit point x_0, appearing counter-clockwise along the boundary of P. Let GE_i be a shortest route, starting and ending at x_0 and visiting the essential extensions in the order $E_i \to E_{i+1} \to \ldots \to E_t \to E_1 \to E_2 \to \ldots \to E_{i-1}$. Let W^* be a shortest closed rectilinear path encircling P, passing from x_0. If x_0 is on the boundary of the polygon then $|GE_1| = |W^*|$. In general, even if x_0 is not on the boundary, we have $|GE_i| \leq 2|W^*|$.*

Based on the above lemma, we can design an online exterior search algorithm similar to greedy online. Essentially, the robot follows a rectilinearization of the taut-thread path, threading all the essential extensions, from x_0 back to x_0. We are now ready to present $O(k)$-competitive algorithm for exploring rectilinear polygons containing at most k rectilinear obstacles.

The algorithm is similar to greedy online. We will learn the boundaries of the objects one by one: For each object O_i, the robot will encircle O_i. Then we pick another object among those whose boundaries are partially unknown, and return to the closest point from where it can see that object. Then the robot continues with the exploration of that object. If we can show that the optimal loop W_i for viewing the obstacle O_i is no longer than the optimal watchman's route W^* for viewing the whole polygon, including the obstacles, this algorithm clearly has an $O(k)$ competitive ratio. We present the proof in the next section.

The algorithm makes use of the frontier f, similar to greedy online. The robot adopts the so called "to the right facing the wall" direction (2RFW), which is clockwise for the interior of the surrounding polygon and counter clockwise for the obstacles. A point that remains obscure is how to pick an initial point to explore an obstacle. In the next section we will show that a path that starts and ends at x_0 is no longer than a constant multiple of the minimal search path length with no constraints. Another technicality is that the rectilinearization of the shortest watchman route around the obstacle is not unique. We get around this by imposing the condition that the path have the maximum overlap with the boundary of the obstacle.

4.4 Analysis

In this section we will prove the claim we made earlier. Namely, we will show that the algorithm outlined in the previous section is $O(k)$ competitive for a rectilinear room with k obstacles. We will need the following variables:

- W^*, Optimal Watchman's route, the shortest rectilinear cycle viewing the boundaries of all objects.
- O_i, obstacle i, $i \in [1 \dots k]$.
- Q_i, the set of all points visible from O_i.
- ρ_i, shortest route viewing all points in Q_i.
- π_i, shortest route viewing all points in Q_i starting and ending at x_0.

The proof proceeds as follows:

1. Among all the routes viewing Q_i, there exits a ρ_i that encloses O_i only.
2. Note that $|\rho_i| \leq |\pi_i| \leq |W^*|$.
3. π_i can be approximated online. The online approximation, $\overline{\pi_i}$ is no longer than a constant multiple of $|W^*|$.
4. if we traverse $\overline{\pi_i},\ i \in [1 \dots k]$, the total distance traversed is $O(k)|W^*|$.

Let d_1 be the shortest distance from x_0 to a point on ρ_i. At any time, when the robot is at x_0, it has seen zero or more objects and the boundaries of other objects are broken into known pieces. Let p be where one point on these pieces becomes visible, such that the distance between x_0 and p is minimum. Let γ_i be the shortest rectilinear path from p back to p, enclosing O_i. Let d_2 be the length of the shortest path from p to ρ_i. Note that point p can be found with the map constructed so far.

We are trying to show that the online approximation of π_i, $\overline{\pi_i}$, is no more than a constant multiple of π_i. Obviously $dist(x_0, p) \leq |\pi_i|$. Our robot goes to p, travels along γ_i and returns. Hence the total distance traveled is bounded by $|\gamma_i| + 2|\pi_i|$.

Clearly, we have $d_2 \leq dist(p, x_0) + d_1 \leq 2|\pi_i|$. (To see why $d_1 \leq |\pi_i|$, first note that ρ_i and π_i must intersect, otherwise one would be encircling the other which contradicts the minimality of both. Then, π_i reaches ρ_i and we have a path from x_0 to a point on ρ_i as a part of π_i, hence this path is no longer than π_i itself.).Thus, $|\gamma_i| \leq 2|d_2| + |\rho_i| \leq 4|\pi_i| + |\rho_i| \leq 5|\pi_i|$. Therefore we have $|\overline{\pi_i}| \leq 7|\pi_i|$.

4.5 A More Realistic Algorithm

In the previous algorithm we assumed that the robot had an infinite range of vision. However, in practice robot's sensors have a finite range. In [1], this situation was modelled by adding a grid to the scene and restricting the robot's motion to the nodes and vertices of the grid. A node in the grid models the vicinity that robot can see at a given point. Now, to explore the environment the robot has to traverse all the nodes and edges of the grid, using as few edge traversals as possible. This is illustrated in figure 13.

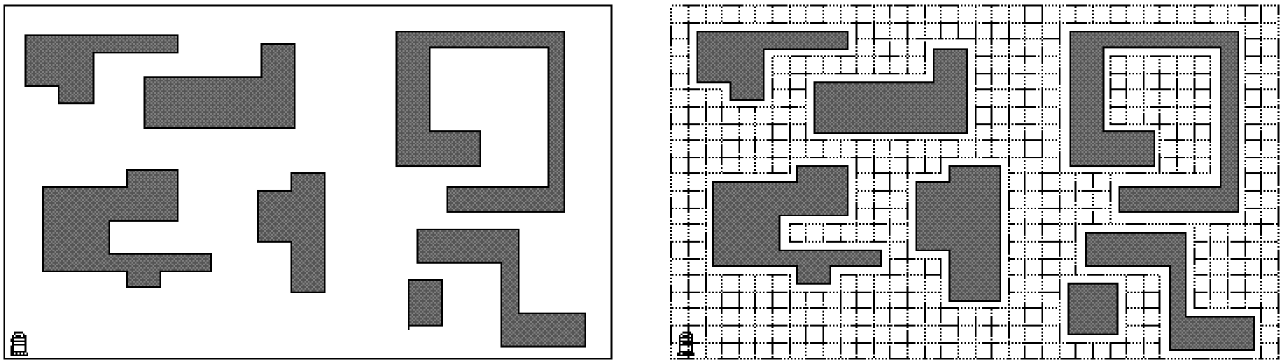

Figure 13: (left) a sample scene. (right) sample scene with a grid. taken from [1]

An additional constraint introduced in this section is the so called *piecemeal constraint*, where the robot has to return to the starting point every so often. In real life, this need can arise when a plane has to return to the base to refuel every now and then, for example. We now present the algorithm.

The algorithm presented in [1] is the generalization of the Ray Algorithm presented in [3], where the obstacles are assumed to be rectangles. This assumption is used to find a path back to s of some maximum length. When exploring grids with arbitrary (orthogonal) obstacles this cannot be satisfied all the time. To overcome this difficulty the algorithm in [1] explores the boundaries of the obstacles efficiently.

Consider a grid graph with arbitrary obstacles and let s be the start node the robot has to relocate to. We define the radius r of the graph to be the maximum of all shortest path distances between s and any other node in the graph. It is assumed that the robot can travel $2(3+\alpha)r$ edges between two consecutive visits to s, so that it can traverse $R = (3+\alpha)r$ edges before moving back to s. Here α is some constant, which will be explained later. Let *dist* represent a variable that gives the number of edges traversed since the last visit to s. We make the following assumptions:

1. the exterior boundary of the grid is a rectangle and no obstacle touches the boundary,
2. starting node s is located at the bottom left corner of the scene,
3. no edge belongs to the boundary of two obstacles.

In the first step the robot traverses the exterior boundary of the scene, by moving clockwise from s. For the moment assume that there exists a procedure *Refuel*, which when called from node x, makes the robot move to s and then back to x. Whenever $dist = R$ the robot calls Refuel.

The actual exploration of the algorithm proceeds in rays that start from $s \in \{x_1, x_2, \ldots, x_n\}$, the vertices of the bottom segment of the scene and proceed in northern direction until they hit the boundary of an obstacle. While exploring vertical rays, the robot moves one edge to the west to explore the horizontal edges. The Ray algorithm follows a depth first strategy, whenever the robot hits a new obstacle O at a node y, while exploring the ray R_i starting at x_i, the robot immediately calls $Map-Obstacles$ described below. The process of shooting rays is illustrated in figure 14.

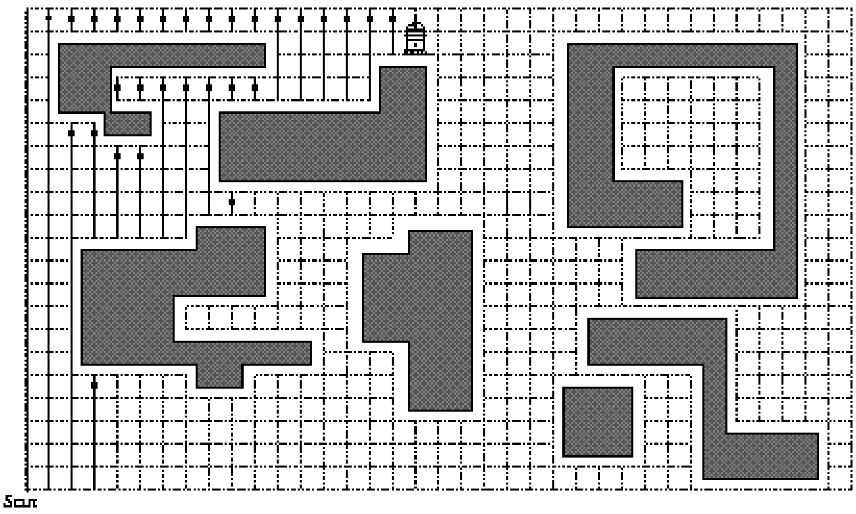

Figure 14: Shooting rays. Taken from [1]

Map-Obstacles: Assume that the robot hits the obstacle O at node x. Initially the robot moves in clockwise direction along O and tries to reach x. If it can reach x while $dist \leq R$ then we are done. In the mean time, the robot maintains a set B of breakpoints that correspond to partially mapped obstacles. Initially x is inserted to B. If the robot cannot reach x while $dist \leq R$, then the node z reached when $dist = R$ is added to B. Then the robot moves back to s, tracing the path it have followed so far. Then it computes a path P from s to z assuming all the obstacles are known. Since there will be some partially mapped obstacles on the way, we must pay attention not to walk into those obstacles. The algorithm proceeds in phases until B is empty. It keeps track of the breakpoints which are the endpoints of explored segments. For an obstacle O, we define the open segment S of O as a maximal sequence of consecutive horizontal edges on the boundary of O such that the interior of O is to the south of S. At the beginning of any phase the robot is given a path P from the most recently inserted breakpoint $b_0 \in B$. Three different cases are possible:

Case 1: The robot hits b_0 or some other $b \in B$.
In this case b_0 or b is deleted from B, and the robot moves along the unexplored boundary of the partially mapped obstacle. If some $b' \in B$ is reached, b' is deleted if no unexplored boundary starts at b'. Otherwise if $dist = R$ the current node is inserted as a breakpoint b_0 into B. In both cases the robot moves back to s and computes a shortest path P from s to the most recent node in B.

Case 2: The robot hits a node y that belongs to the unexplored boundary of an obstacle.
y is inserted into B, and a boolean flag new is set. The robot moves along the unexplored boundary. If $dist = R$ and no other break point was reached, the current node is inserted into B and the robot moves back to S. If y is reached, then the robot successfully mapped the new obstacle, so y is deleted from B and the robot computes a new shortest path from y to b_0. If the robot reaches some other $b \neq y, b \in B$, then b is deleted from B and P is the path just traversed between y and b. In the next phase the robot tries to reach y and explore new boundary edges starting from y. In any case new is reset.

Case 3: If the robot runs out of fuel (dist = R) on the way, the robot moves back to s and computes a new shortest path from s to b_0.

If in the end B is empty but the robot is not located on x, we insert x into B and the robot computes a shortest path from the current node to x, following the same exploration strategy. Case 1 and Case 2 are illustrated in figure 15.

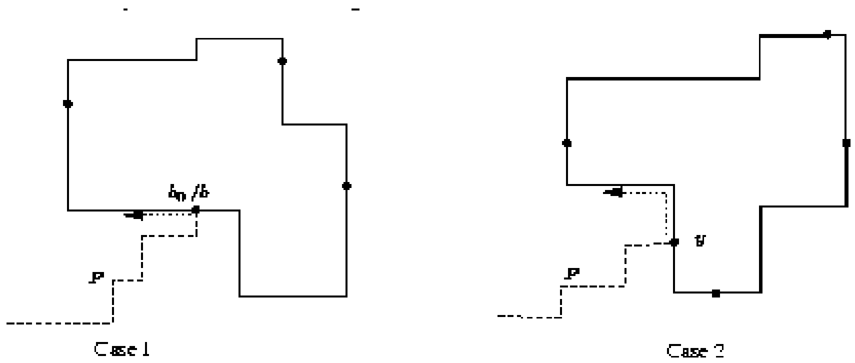

Figure 15: Cases 1 and 2 of Map-Obstacles, partially observed segments are the bold lines.taken from [1]

When Map-Obstacles terminates, the obstacle O and all obstacles hit during the execution of the procedure are mapped fully and the robot is located at y, the point where Map-Obstacles was called initially. Then, the robot backtracks to x_i and then shoots another ray starting from x_{i+1}.
Before we move on to the analysis we present the complete algorithm using a top-down approach:

Algorithm Ray:

1. move along the exterior boundary of the scene until s is reached again. Execute Refuel whenever dist = R.
2. S ← the bottom segment of the boundary.
3. Shoot-Rays(S);

Shoot-Rays(S):

1. Initially,
    x ← the current node where the robot is located.
    $\{x_1, \ldots, x_k\}$ ← vertices of S.
2. for i = 1 to k do
3.   Move to x_i
4.   Move on a vertical ray R_i in northern direction until the boundary or some obstacle is hit.
    Execute Refuel whenever dist = R.
5.   y ← current node.
6.   if a new obstacle O was hit then
7.   Map-Obstacles
8. $\{S_1, \ldots, S_l\}$ ← the open segments of O in cw order wrt y.
9.   for j = 1 to l do
10.   Move in cw direction along O to S_j. Execute Refuel when $dist = R$.
11.   Shoot rays(S_j).
12.   Move to y along the boundary of O. Execute Refuel when $dist = R$.
13.   Move to x_i following the ray R_i backwards. Execute Refuel when $dist = R$.

Procedure Refuel:

1. x ← the current node
2. Move back to s, using the path that is traversed since the last visit.
3. while x is not reached do
4.   P←shortest path from current node to x,assuming all obstacles are known.
5.   Follow P until a new obstacle is reached or dist = R or x is reached.
6.   If some new obstacle is hit then
7.   MapObstacles
8.   If dist=R then
9.   Move back to s

Procedure MapObstacles :

1. Initially,
    x ← the current node
    B ← $\{x\}$
    new ← true
    y ← x
2. while $B \neq \emptyset$ do
3.   if dist < R then
4.   Move along the unexplored boundary until some $b \in B$ is reached or dist = R.
5.   if some $b \in B$ is reached then
6.   if new = false then
7.   Move back to s;
8.   else if new = true and b =y then
9.   new←false;
10.   if no unexplored boundary starts at b then delete b from B;
11.   if dist = R then
12.   if unexplored boundary starts at current node then
13.   Insert current node into b
            new←false

```
14.          move back to s.
15.    if B=∅ and current node ≠ x then
16.          B ← x
17.    if B≠ ∅ then
18.          if new=true then
19.                P ← shortest path from current node
                   to most recent node in B that follows the boundary just explored.
                   new ← false
20.          else
21.                P ← shortest path from current node
                   to most recent node in B assuming all obstacles are known.
22.          Move along P until some b∈ B was reached or some unknown boundary is hit or dist = R
23.          if some b∈ B was reached and B was incident to an explored boundary edge then
24.                delete b from B.
25.          else if some unknown boundary is hit at y then
26.                insert y into B;
                   new ← true.
```

4.6 Analysis of the Algorithm

4.6.1 Correctness

In order to prove the correctness of the algorithm we give the following lemma without proof:

Lemma 4.10 *Assume that at the beginning of an execution of Map-Obstacles all previously hit obstacles are mapped fully. Then the following two statements hold at the end of the execution*

1. *The obstacles hit at the beginning and during the execution are mapped fully.*
2. *The robot is located at the node where the execution started.*

By the above lemma, all the moves during the ray algorithm are well defined. All the nodes and edges on the boundary of the obstacles are explored, since every obstacle must be hit when the robot traverses in vertical rays. Any other vertical edge lies on a vertical ray, thus is explored. Horizontal edges are explored when the robot moves along vertical rays. Therefore, at the end of the execution of Ray, the entire scene must be explored.

4.6.2 Efficiency

Let N be the total number of visits to s during the execution. Let Q_i be the path traveled by the robot after the i^{th} visit to s, and before it starts its trip back to S. In order to show that the complexity of the algorithm is $O(|E|)$ we will show that $\sum_{i=1}^{N}|Q_i| \in O(|E|)$, since the total number of edge traversals is bounded by $2\sum_{i=1}^{N}|Q_i|$.

An edge is called fresh (1) if it is traversed as a new edge and explored after traversal or (2) if it is traversed in a call of Shoot-Rays excluding the executions of Refuel and Map-Obstacles. All the edges traversed on vertical rays or when moving around obstacles in lines 10-12 of Shoot-Rays are fresh. Here is the outline of the proof:

Lemma 4.11 *Suppose that the robot is located at a node y and then repeatedly executes lines 4-9 of Refuel. Whenever it has traversed 2r+l edges, $l > 0$, since the visit to y and has not yet moved back to s, it has traversed at least $\frac{l}{2}$ fresh edges. .*

Proof sketch: In line 4 of Refuel, the robot computes an initial shortest path P from its current position y to the target location x (i.e where it started its journey back to s). By definition $|P| \leq 2r$. Thus for the robot to travel 2r+l edges it must have hit an obstacle. We know that less than $|P|$ edges were traversed before the robot hit the first obstacle hit, O. All the edges on

the obstacles are fresh, therefore the fresh edges traversed is at least l[1].
In the case where the robot has hit k obstacles we define y_j to be the point where the j^{th} obstacle, O_j was hit; P_j to be the path calculated from y_j to x; P'_j to be part of P between y and y_j, $|O_j|$ to be the number of boundary edges on $|O_j|$. $P'_0 = \emptyset$ and $P_0 = P$. If we set $|O_0| = 0$ it is possible to prove by induction that:

$$|P'_j| + |P_j| \leq |P| + 2\sum_{i<j} |O_i| + |O_j| \ \ holds\ for\ j \in [0, \ldots, k].$$

Using the result above, we know that the robot has traversed $2r + l \leq |P'_k| + |P_k| \leq |P| + 2\sum_{j=1}^{k} |O_j| \leq 2r + 2\sum_{j=1}^{k} |O_j|$ edges and therefore $l \leq \sum_{j=1}^{k} |O_j|$. Since all the edges on the boundaries are fresh, the desired bound follows.
The only case left is when the robot has already explored k obstacles and it is exploring O_{k+1}. If the robot has traversed 2r edges on the way to O_{k+1} we are done, since all the edges on O_{k+1} are fresh. Assume it has traversed $2r + l'$, $l' > 0$ edges so far. This means it has traversed $\frac{l'}{2}$ fresh edges since it has seen $2r + l' \leq |P| + 2\sum_{j=1}^{k} |O_j|$ edges, $\sum_{j=1}^{k} |O_j|$ of which was fresh. Since all the edges on the boundary of O_{k+1} are fresh the total number of fresh edges on the path is $(2r + l) - (2r + l') + \frac{l'}{2} = l - \frac{l'}{2} \geq \frac{l}{2}$ since $l \geq l'$.
Assume that the robot hits the obstacle O at point y. Let y_l be the nearest explored node on the boundary of O in clockwise direction, and y_r be the nearest explored node on the boundary of O in counter-clockwise direction. It is possible to have $y = y_l$ and/or y_r.
In order to prove a similar result for Map-Obstacles, we will need the following proposition:

Proposition 4.12 *Suppose that the robot hits a partially mapped obstacle O at y during Map-Obstacles.The following hold while* $dist \leq R$.

1. *if* $y = y_l$ *or* $y = y_r$*, then the robot moves from y along the unexplored boundary of O until the second of the two breakpoints is hit.*
2. *if* $y \neq y_l$ *and* $y \neq y_r$*, then the robot moves in one direction along the unexplored boundary of O, until either* y_l *or* y_r *is hit. It then returns to y and moves along unexplored boundary to the second break-point.*

When all edges between y_l *and* y_r *are explored, the robot immediately moves back to s.*

This proposition can be verified directly by the code. Part 1 can be verified by observing lines 4,7,17 and 19 and part 2 can be verified using lines 26, 4, 19,22 and 7 of Map-Obstacles.

Lemma 4.13 *Suppose that the robot is located at s and then executes code of Map-Obstacles. Whenever the robot has traversed r+l edges,* $l > 0$*, since the last visit to s and has to yet moved back to s, it has traversed* $\frac{l}{2}$ *fresh edges.*

Proof Sketch: Assume that the robot is located at s, and b is the most recent break-point. On the way to b, if the robot does not hit a partially mapped obstacle, lemma 4.13 can be proven in the same way as lemma 4.13. Assume that robot hits partially mapped O' at y, then y must be between y_l and y_r by proposition 4.12. If $y = y_l$ or $y = y_r$ all the edges traversed on the boundary of O'_i are fresh. If $y \neq y_l$ and $y \neq y_r$ then at least half of the edges on O' are fresh. Let $\mathcal{O}$ be the set of all obstacles hit before O' and P be the initial path from s to b. The robot must have travelled $|P| + 2\sum_{O\in\mathcal{O}} |O| \leq r + 2\sum_{O\in\mathcal{O}} |O|$ edges. Thus the total number of fresh edges is at least $\frac{1}{2}(r + l - (r + 2\sum_{O\in\mathcal{O}} |O|) + \sum_{O\in\mathcal{O}} |O| = \frac{l}{2}$.
From the above lemmata we get:

Lemma 4.14 *If a path* Q_i *traverses* $R = (3 + \alpha)r$ *edges, then at least* $\frac{\alpha r}{2}$ *these edges are fresh.*

The proof of the above lemma follows from lemmata 4.11 and 4.13. There are three different cases depending on the procedure in which the visit to s at the beginning of Q_i occurs. We refer the reader to the paper [1] for the details.

[1] $2r + l - |P| \geq l$

A fragment is a maximal sequence of consecutive edges that were explored successively on one of Q_i. Thus, we can associate any fragment to a unique Q_i.

Lemma 4.15 *Assume O has $k \geq 2$ fragments. There exists at least $\lfloor \frac{k}{2} \rfloor$ fragments whose Q_i traverse $R = (3+\alpha)r$ edges.*

Lemma 4.16 *the number of Q_i that traverse R edges $\geq$ the number of paths that traverse less than R edges.*

The only case when a path has less than R edges is when it hits a partially mapped obstacle as in proposition 4.12. Hence, any path that traverses less than R edges ends on the boundary of an obstacle and associated fragment belongs to an explored segment with at least two fragments.

Theorem 4.17 *Total number of edge traversals is $O(|E|)$.*

Proof: In the algorithm Ray, every fresh edge is traversed at most four times:

1. at least once for the exploration
2. at most three times in ShootRays: Twice if it backtracks, and once on the edge boundary.

Hence the algorithm traverses $4|E|$ fresh edges at most. By lemma 4.13 this means there are at most $\frac{2 \cdot 4|E|}{\alpha r}$ paths Q_i that traverse R edges. By lemma 4.16 the paths Q_i that traverse less than R edges is also bounded by the same number. Finally $\sum_{i=1}^{N} |Q_i| \leq \frac{16|E|}{\alpha r}(3+\alpha)r \in O(|E|)$.

5 A competitive algorithm for the rectilinear case?

In the previous section we presented an $O(k)$ competitive algorithm where k is the number of obstacles in the scene. This gives a constant ratio when the number of obstacles is bounded. In [7], Deng et al. conjectured the existence of a competitive algorithm even when the number of obstacles is unbounded. Unfortunately, the existence of a competitive algorithm for a rectilinear environment with an arbitrary number of polygons can be constructively refuted. In this section, we present the construction in [1] to disprove a competitive ratio and we show that any deterministic algorithm that explores a rectilinear environment with arbitrary number of obstacles is at least $\Omega(\sqrt{n})$ competitive.

The construction uses k-combs illustrated in figure 16. Following variables are used in the construction:

5.1 Variables

- **n:** the number of obstacles is $\Theta(n)$
- **k:** total number of stages. $k = \lfloor \sqrt{\frac{n}{2}} \rfloor$.
- **k-comb:** the generic obstacle with k spikes and k-1 bases
- w_b**:** width of base rectangles $w_b = 1$.
- ϵ**:** the distance between the k-comb of two consecutive stages. $\epsilon = \frac{1}{2}(2k)^{-k}$.
- ϵ'**:**the distance between a spike and a base. $\epsilon' = \frac{\epsilon}{2k}$.
- H_i**:** The height available for the k-comb at stage i. $H_1 = k$ and $H_i = \frac{H_{i-1}}{2k} = \frac{H_1}{(2k)^{i-1}}$.
- W_i**:** The width available for the k-comb at stage i. $W_1 = 2k$.
- h_i**:** The height of the base rectangle at stage i. $h_i = \frac{H_i}{2k} + 2\epsilon$.

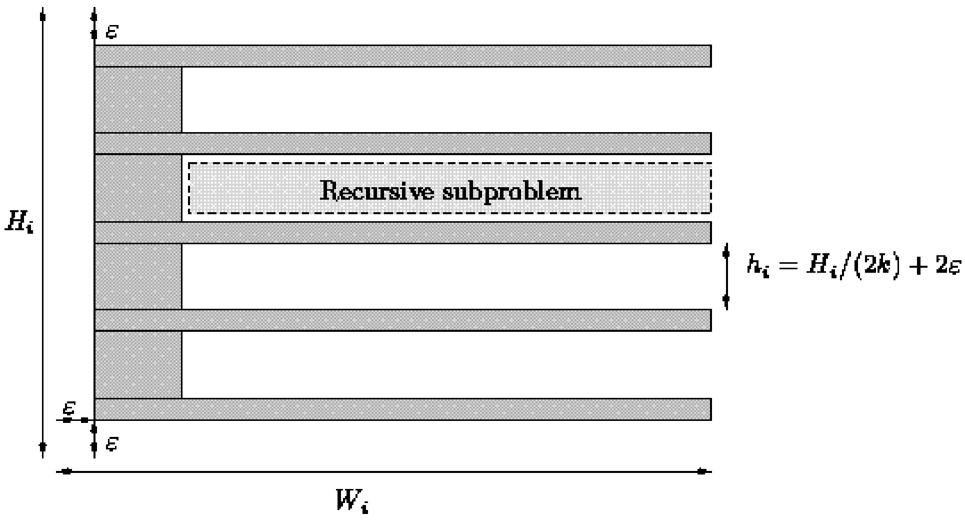

Figure 16: A k-comb of stage i with k spikes and k-1 base rectangles. Taken from [1]

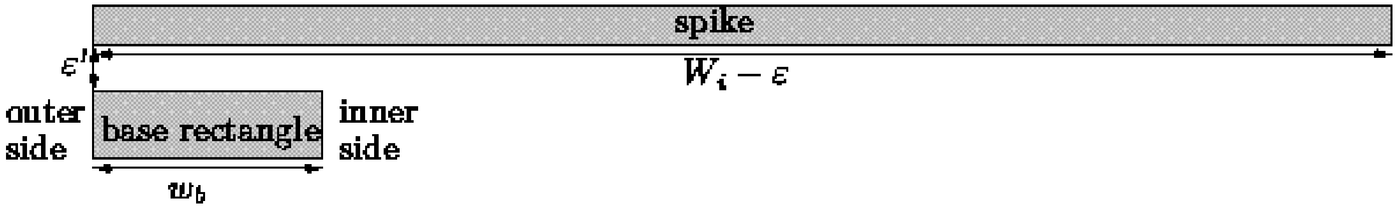

Figure 17: A spike and a base rectangle. Taken from [1]

5.2 Construction

We set the following variables such that

$$W_i = 2k - (w_b + \epsilon)(i - 1) \geq k \tag{1}$$

The height of a spike is then

$$\frac{H_i' - 2\epsilon}{k} \tag{2}$$

$$where\ H_i' = H_i - (k - 1)(\frac{H_i}{2k} + 2(\epsilon + \epsilon')) \tag{3}$$

5.3 Placement

For the deterministic exploration algorithm A, we place the k-combs such that the robot has to explore the inner sides of all the base rectangles. The k-comb of stage i is placed such that the robot faces the outer side of the base-rectangles (i.e in stage i we put the mirror image of the comb at stage i-1 with appropriate resizing). Therefore, to find the k-comb of stage i, the robot has to explore the inner sides of the base rectangles.

Given this construction we are now ready to take a look at the strategies.

5.4 The Optimal Algorithm OPT

Assume the robot is initially in the lower left corner. OPT moves in the following way:

- Move H_i upwards along the left side.
- if $i < k$ move to the left corner of the k-comb of next stage, otherwise move to the right of the k-comb.
- Move up and down again along the right side.

If $i < k$, L^i_{OPT} the distance travelled in stage i is:

$$L^i_{OPT} \leq 2H_i + w_b + \epsilon + 2H_i = 4H_i + w_b + \epsilon \tag{4}$$

If $i < k$, since the robot moves right in addition

$$L^k_{OPT} \leq 4H_k + 2k - (k-1)(w_b + \epsilon) \tag{5}$$

Summing up we get,

$$L_{OPT} = \sum_{i=1}^{k-1} L^i_{OPT} + L^k_{OPT} \leq 10k \tag{6}$$

5.5 The online algorithm

For the online algorithm, we are going to show that the distance travelled in the horizontal direction is $O(k^2)$. The main difference between the online version and the offline version is that in the former one the robot does not know where the next stage comb lies. Therefore, it has to search for it. When the robot is at a lower corner of the stage i, it has two options:

- Explore the inner side of the base rectangles: This is illustrated in figure 18. The robot has to move to a distance of ϵ from the inner side of the base to see the next comb. Therefore the robot travels a distance of $2(w_b - \epsilon)$ for k-1 rectangles.
- Change sides : In this case the robot travels a distance of $W_i \geq k$.

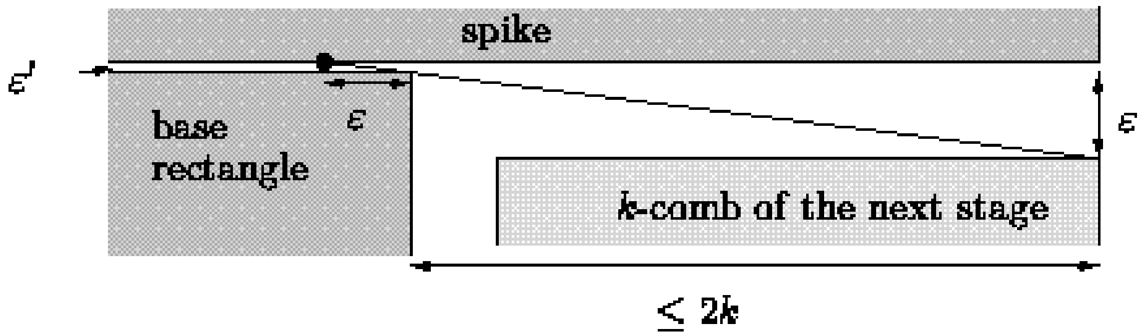

Figure 18: Taken from [1]

Thus, the total length travelled is $\sum_{i=1}^{k} k = k^2$. Therefore the ratio is $O(\frac{k^2}{10k}) = O(k) = O(\sqrt{n})$. We summarize this result in the following theorem:

Theorem 5.1 *Let A be a deterministic online algorithm for exploring two-dimensional scenes with n rectangles. If A is c-competitive, then* $c = \Omega(\sqrt{n})$.

5.6 A lower bound for the randomized algorithms

We will use a randomization of the construction of the previous section to present a lower bound for the randomized algorithms. The only difference is the placement of the k-combs. In each stage, we will place the k-comb so that its outer side is to the left with probability $\frac{1}{2}$ and it is to the right with probability $\frac{1}{2}$. We also choose the placement of the k-comb randomly: each of the k-1 configurations is equally likely.

The upper bound for OPT does not change. For the online algorithm, at any stage the robot is located at the opposite side of the base rectangles with probability 1/2. In this case there is no cost in the x direction. In the remaining case, the robot can either change sides, traversing a length of at least k, or it can search for the next k-comb. Then the expected number of the base rectangles it has to explore is k/2. Then the total length reduces by a factor of 4 but it is still $\Omega(k^2)$. We are ready to prove the following theorem:

Theorem 5.2 *Let R be a randomized online algorithm for exploring two-dimensional scenes with n rectangles. If R is c-competitive, then $c = \Omega(\sqrt{n})$.*

Proof: We choose the input distribution as described above. Then the desired ratio follows from Yao's minimax principle (see appendix).

6 Exploring a Simple Polygon

In the previous section, we designed a competitive strategy for rectilinear polygons which visits the extensions in clockwise order. Basically, this strategy worked in the rectilinear case since two cuts can only cross perpendicularly. Unfortunately we can not make use of this strategy in the general case: our robot can move back and forth between right and left reflex vertices as in figure 19. In general, when we first see a vertex, we can not go straight to it, because its cut may be passing from a point very close to our current position.

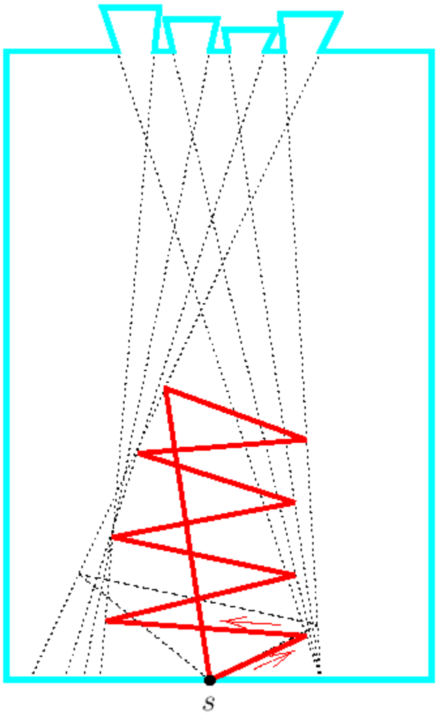

Figure 19: The strategy for rectilinear polygons does not work in the general case. Taken from [10]

Figure 19 suggests that an efficient strategy must visit right and left reflex vertices separately. This is indeed the strategy in [9] where the authors have presented a 133-competitive strategy.

At the heart of their algorithm lies the concept of a room: In a room it is possible to have only one switch from a right vertex to a left one or vice versa. The analysis of this algorithm is somewhat obscure due to *the difficulty in establishing sharp length estimates for robot paths of complex structure, and in relating them to the optimum watchman route[10].*

In this section, we will present the improved version of this algorithm described in [10]. The algorithm makes use of a new geometric structure called the angle hull. Hoffman et al. devoted a separate technical report for the angle hull[11]. Here we will define it and list its relevant properties.

Definition 6.1 *Let D be a simple polygon contained in another simple polygon, P. Then the angle hull, $\mathcal{AH}(D)$, of D consists of all points in P that can see two points of D at an angle of 90^o.*

We can think of $\mathcal{AH}(D)$ as the path traversed by a photographer who wants to shoot D with a 90^o lens but does not want P to appear in any image(see figure 20). We'll take the following as a theorem.

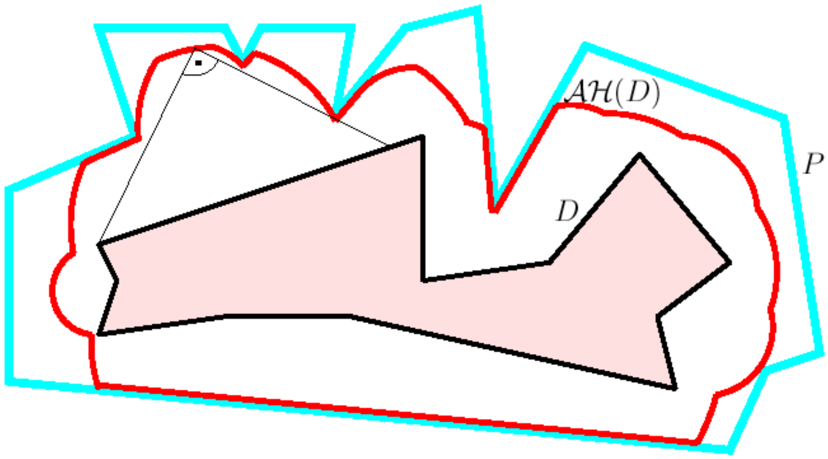

Figure 20: The angle hull. Details can be found in [11]. Image taken from [10].

Theorem 6.2 *Let P be a simple polygon containing a simple polygon D. The arc length of the boundary of the angle hull, $\mathcal{AH}(D)$, with respect to P is less than 2 times the length of D's boundary. This bound is tight.*

Before we give a bottom up presentation of the algorithm we present some essential concepts. In the following, let P be a polygon and s, the starting point, be a point on its boundary. The shortest path tree of s is the tree whose internal nodes are the reflex vertices of P and it contains all the shortest paths from s to the vertices. On the way from s to a vertex v, if the path makes a right turn at an internal node, then it is called a right reflex vertex. Left reflex vertices are defined accordingly. In order to distinguish between seeing a vertex and visiting its extension we call a vertex *discovered* after it has been visible at least once from the robot's current position. A vertex is *unexplored* before we cross its cut and it is *fully explored* afterwards.

When a robot sees a vertex r from its current position p, instead of walking straight to it, it walks along the smallest circle that contains p and r [2]. Although this path may be as long as $\frac{\pi}{2}$ times the Euclidean distance from p to r, due to Thales' theorem we arrive the cut at the point

[2]we refer to this circle as the circle spanned by p and r from here on and use the notation circ(p,r)

that is closest to p. Note that we can not walk directly to the extension, for we do not know its orientation (we have not seen the associated edge).

The algorithm first divides the polygon into group of vertices. Each group starts from a stage point, which is basically a vertex of the shortest path tree of s such that the shortest path from s to any vertex of a group leads through the group's stage point. The stage point of a right group is always a left vertex. We define the starting point to be the first stage point. The algorithm then visits the first vertex of the group[3] using the algorithm described next.

6.1 Exploring a single vertex

In this section we present the subroutine ExploreRightVertex that explores a single vertex[4]. The algorithm uses the following variables :

- **BasePoint:** The robot's initial position when it enters ExploreRightVertex.
- **TargetList:** A list of right vertices whose shortest path from the base point makes only right turns, sorted in clockwise order along the boundary, that have already been discovered but not explored.
- **ToDoList:** This list is used to keep track of vertices discovered on the way to another vertex.
- **CP:** Current position of the robot.
- **Target:** The first vertex of TargetList.
- **Back:** The last vertex on the shortest path from the base point to CP, excluding CP.

The robot tries to explore Target, it also keeps a long term agenda ToDoList. Note that ExploreRightVertex may add but not remove from ToDoList. On the way to target several things can happen:

- If a new right vertex r is discovered, we check if the shortest path from the current stage point to r contains any left turns. If not, then we insert r to the TargetList. If r comes before the Target then we start exploring r, i.e we start moving along circ(Back,r).
- If the robot loses sight of the BasePoint, then it starts moving along circ(Target, Back).
- If the robot crosses a right vertex, other than the Target then that vertex is removed from the TargetList, since it is explored anyway.
- If the robot hits the boundary, the robot follows the boundary until it is possible to walk along the circle again.
- If the robot is about to lose sight of Target, it walks straight to the blocking vertex, and continues there from a circular path.

In general, when a right vertex is explored all of its children that have no let children on the shortest path tree are discovered. When we exit from ExploreRightVertex, the robot is done with the exploration of the Target -it removes it from the ToDoList-, and it has inserted the right vertices having a left child to the ToDoList. It is now situated on the Target's cut.

The reason we are ignoring the vertices that have left turns may not be clear at first. Since our intention is to find a competitive algorithm, this strategy allows us to estimate the length of the path travelled:

Lemma 6.3 *Suppose procedure ExploreRightVertex terminates with the robot reaching the cut of the vertex r_1 at point c. Then the robot's path is a part of the boundary of the angle hull $\mathcal{AH}(R)$ of the shortest path, R, from the base point r_1, except for straight line segments leading to blocking vertices. Furthermore, point c is the point of the cut closest to the base point.*

[3] clockwise along the boundary

[4] Since the intention of the algorithm is to visit left and right vertices separately, there are separate procedures for exploring a right vertex and a left vertex. Here we present the procedures for the right vertices, left ones are obtained by exchanging left and right, clockwise and counterclockwise

In addition, the straight line segments mentioned in lemma 6.3 are in fact circular arcs in the angle hull, therefore the length of the path traversed is bounded by the length of the angle hull. Using theorem 6.2 we conclude that the path traversed in ExploreRightVertex is not longer than twice the length of the shortest path.

6.2 Exploring Group of Vertices

In this section we will specify how the groups of vertices are determined and we will present the procedure ExploreRightGroup that explores a group. We will then take a look at some properties of ExploreRightGroup.

In the previous section we mentioned that the procedure ExploreRightVertex may add vertices to the TargetList along the way. Assume that the current stage point is s. When we call ExploreRightVertex repeatedly, till the TargetList becomes empty, we see that all right vertices initially present in TargetList have been explored, along with their purely right descendants in the shortest path tree of s. This set of vertices is called a group.

The robot explores a group as follows. It repeatedly calls ExploreRightVertex. On return, it walks along the cut it has just explored to the point closest to the stage point. It uses the point it has just arrived at as the base point in the next execution of the ExploreRightVertex. When TargetList becomes empty, the robot walks back to the stage point.

We now present some properties of ExploreRightGroup:

Lemma 6.4 *Let $b_1, \dots, b_m$ be the base points generated in m consecutive calls of ExploreRightVertex. The shortest paths from the stage point to $b_1, \dots, b_m$ are in clockwise order.*

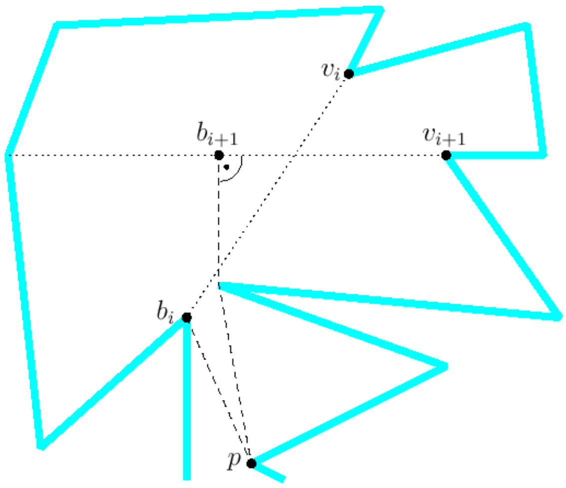

Figure 21: The shortest path to b_i runs to the left of the shortest path b_{i+1}. Taken from [10]

Proof: Consider the situation in figure 21. Let $b_1, \dots, b_m$ be situated on the extensions of $v_1, \dots v_m$. Assume that the robot is located at p. We are trying to show that, the shortest path to b_i runs to the left of the shortest path b_{i+1}. First note that v_i must appear in clockwise order by design. In addition, both p and the stage point must lie below both v_i and v_{i+1}, since we haven't explored them yet. The lemma follows from the observation that b_i must be below b_{i+1}, because b_{i+1} is still unexplored when we reach b_i.

Lemma 6.5 *The robot's path between two consecutive base points is at most 3 times as long as the shortest path.*

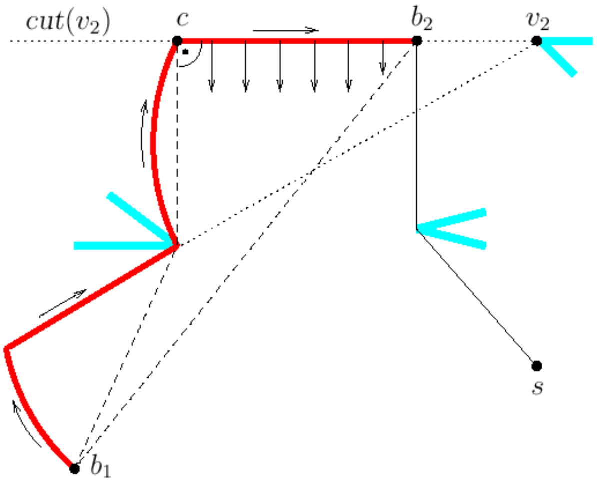

Figure 22: The generic figure for the proof of lemma 6.5. We use the notation cut(v_2) for the extension of the side that contains v_2. Taken from [10].

Proof: The situation is illustrated in figure 22 where we arrive cut(v_2) at c. By lemma 6.4, we know that the shortest path to b_1 must be to the left of the shortest path to b_2 and b_1 must be below the cut of v_2. We also know that the distance from b_1 to c is no longer than twice the length of the shortest path from b_1 to c. The walk from c to b_2 can be projected onto the shortest path, so it cannot be longer than that. Note that this section of the walk cannot intersect the first part, because we are moving to the point closest to s. Therefore we conclude that the total distance is no longer that 3 times the shortest path.

Lemma 6.6 *The length of the path caused by a call to ExploreRightGroup does not exceed* $3\sqrt{2}$ *times the perimeter of the relative convex hull,* $\mathcal{RCH}$*, of the base points visited*[5]*.*

Proof Assume that one execution of ExploreRightVertex starts from the stage point sp and let $\{sp = b_0, b_1, \ldots, b_{m-1}, b_m\}$ be the basepoints. If all all $b_1, \ldots, b_m$ are on the $\mathcal{RCH}$ then we are done, due to lemma 6.5. Suppose for some $i \leq k-2$, b_i and b_k are on the $\mathcal{RCH}$ and $b_j,\ i < j < k$ the ones in between, are not. Note that the shortest path from sp to each cut must have a right angle. In addition, the cut of b_j must pass above b_{j-1}, since it is explored afterwards. The situation is illustrated in figure 23. Whenever the path makes a left turn at v, we move v to v' where v' is chosen such that it is on the shortest path from sp to the successive basepoint and the left turn becomes a right angle. For example, b_{j+1} in figure 23 is replaced by b'_{j+1}. Clearly this new path is longer than the previous one. This leads to a path where all left turns are right angles. Finally we replace this resulting path with one that has only one left turn; b_i, b'_j, b'_{j+1} in the figure. The ratio in this case becomes $\frac{3a+3a}{a\sqrt{2}} = 3\sqrt{2}$ where a is the larger of the sides adjacent to the 90^o angle. Note that this path (of length 6a) is even longer than the distance travelled by ExploreRightGroup.

Lemma 6.7 *All base points are contained in the angle hull* $\mathcal{AH}(W_{OPT})$*.*

Proof The Optimum Watchman Route W_{OPT} visits all the essential cuts. Consider the last edge E of the shortest path from s to this cut. This edge meets the cut at the basepoint b, by definition. There are two possibilities:

[5]The relative convex hull of a subset D of a polygon P is the smallest subset of P that contains D and for any two points in D, the shortest path in P connecting them.

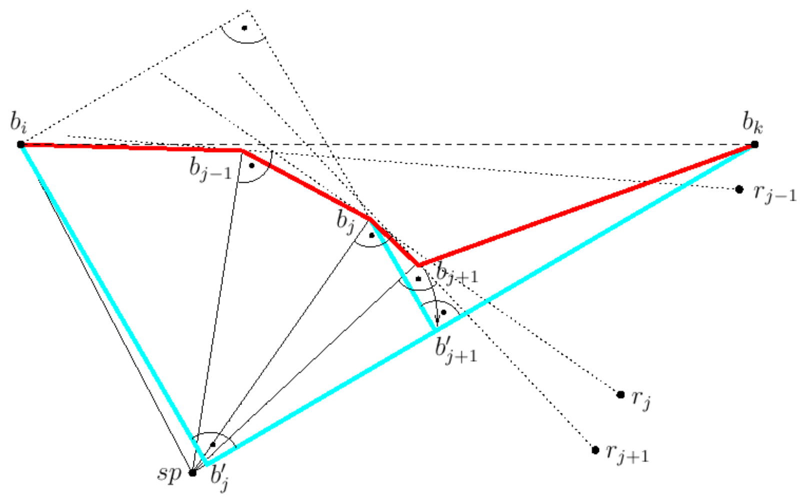

Figure 23: The generic figure for the proof of lemma 6.6. Taken from [10].

If E is orthogonal, then there is a right angle at b, whose one side goes along the cut and touches W_{OPT}, and the other side K extends to E. Now either K intersect s and touches W_{OPT} there, or K separates s and the cut and therefore must be touched by W_{OPT}.

If E is not orthogonal, then b is one endpoint of the cut. By similar arguments there must an angle greater than 90^o both of whose sides touches W_{OPT}.

6.3 Exploring the Polygon

In this section, we present the complete algorithm and its analysis. We'll first present the pseudocode and some of its properties:

procedure ExplorePolygon(P,s)
  ExploreRightGroup(Targetlist={cw list of all vertices visible from s}, ToDoList);
  TargetList ← all left children of the vertices of the ToDoList;
  Add all left vertices visible from s into TargetList and sort cw;
  ExploreLeftGroupRec(TargetList);
end ExplorePolygon

procedure ExploreRightGroupRec(TargetList)
  ExploreRightGroup(TargetList, ToDoList);
  Clean up ToDoList:
    retain only those right vertices in ToDoList, which are highest up in the shortest path tree;
  for all vertices v of ToDoList in cw order do
    walk on the shortest path to v /* connect stage points */
    ExploreLeftGroupRec({all known left descendants of v in ccw order});
end ExploreRightGroupRec

procedure ExploreLeftGroupRec(TargetList)
  ExploreLeftGroup(TargetList, ToDoList);
  Clean up ToDoList:
    retain only those left vertices in ToDoList which are highest up in the shortest path tree;
  for all vertices v of ToDoList in cw order do
    walk on the shortest path to v /* connect stage points */

ExploreRightGroupRec({all known right descendants of v in cw order});
end ExploreLeftGroupRec

ExploreRightGroupRec works in three steps:

1. The first ExploreRightGroup takes the TargetList and fills up the ToDoList.
2. ToDoList -which contains all purely right descendants of the stage point- needs to be cleaned up, since some of these are right descendants of others. Only the highest ones are retained. These will be future stage points, therefore we assign them the list of left descendants found out in ExploreRightVertex and ExploreRightGroup.
3. Final step is to visit those (left) descendants in clockwise order.

To analyze the algorithm, we categorize the base points as follows:

The set of basepoints generated in the first line of ExplorePolygon are category 0. For others, let i be the recursion depth of the function where a basepoint b is generated. Then b is of category (i mod 3). This way all calls of ExploreRightGroup have even level and all calls of ExploreLeftGroup have odd level.

The key observation leading to an estimate of the path is the following:

Lemma 6.8 *The relative convex hulls of two sets of basepoints of the same category are mutually invisible, with a possible exception of the stage points.*

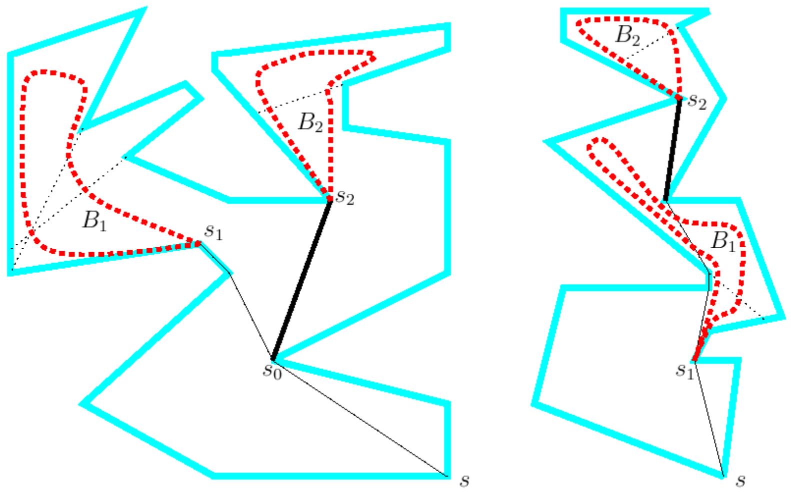

Figure 24: B_1 and B_2 are mutually invisible. Taken from [10].

Proof: Let B_1 and B_2 be two sets of basepoints of the same category. Consider the shortest paths from s to their corresponding stage points s_1 and s_2. Since the difference of levels is 3 (or 0), s_1 and s_2 are of different types and parent stage of s_2 cannot be a direct descendant of s_1 and vice versa. There are two possibilities, illustrated in figure 24.

If s_1 is not on the shortest path from s to s_2, there must be a vertex s_0 where the shortest paths separate. This is illustated in the left picture in figure 24, where we assume the clockwise order $\{s_0, s_1, s_2\}$. In addition, we assume that s_2 is a left vertex without loss of generality. Note that the shortest path from s_0 to s_2 is invisible from any point in B_2, and this shortest path separates B_2 from B_1. They are therefore mutually invisible, except for s_1 and s_2. This argument may be extended to convex hulls as well.

Otherwise assume s_1 lies on the shortest path from s to s_2. Then the difference of the recursion levels is at least three, as in the right picture in figure 24. Similar to the previous case the path from s_2 to its parent is not visible to any basepoint in B_2 and the lemma follows from the same argument. ∎

As a consequence $per(\mathcal{RCH}(B_1)) + per(\mathcal{RCH}(B_2)) \leq per(\mathcal{RCH}(B_1 \cup B_2))$ where per denotes the perimeter.

By lemma 6.7 we know that all basepoints are contained in the angle hull of W_{OPT}, hence the perimeter of their relative convex hull is shorter than $per(\mathcal{RCH}(W_{OPT}))$. On the other hand $per(\mathcal{RCH}(W_{OPT}))$ is not longer than its angle hull. So using lemmata 6.6 and 6.8 we conclude that the path length caused by all calls in one category is no longer that $6\sqrt{2}$ times the length of W_{OPT}. Since we have three categories we travel $3 \cdot 6\sqrt{2}$ times the length of W_{OPT}. The path we take in order to connect the stage points can not be longer than $|W_{OPT}|$ since those points are clockwise along the path and W_{OPT} visits them in the same order as well. Therefore the total length travelled is bounded by $(3 \cdot 6\sqrt{2} + 1)|W_{OPT}| \leq 26.5|W_{OPT}|$

7 Conclusion and Future Research

In this report, we presented the current status of the polygon exploration problem. For rectilinear polygons, in the absence of obstacles there exists an optimal exploration algorithm. We have shown that no exploration algorithm can reach a constant competitive ratio, in the presence of an arbitrary number of obstacles. The general case when the environment is a simple polygon with no holes was also solved. However, the competitive ratio is high (26.5).

One of the open problems in this field is to find a lower bound for the general case. It is also still a challenging problem to find a competitive algorithm with a low competitive ratio. The performance analysis of various standard approaches(e.g depth-first) can be an interesting problem.

In three dimensions, even the shortest path problem becomes hard. It is also not clear how to extend the algorithms presented to higher dimensions. Therefore exploration in 3D remains a challenging problem.

We believe that exploration problem can be solved efficiently with a better representation. In addition, we may not have infinite line of sight or omnidirectional vision in practice, due to the limitations of the sensors. Such a representation should be able to handle situations like limited visibility and limited amount of memory. Our intuition is that quadtrees are good candidates for such a representation and we are planning to investigate this problem next.

A Yao's Technique

In this section we introduce Yao's technique to prove performance lower bounds on approximation algorithms. Our treatment is essentially a summary of the presentation in [13].

Payoff Matrices are standard representations of games in game theory. In general, any two person, zero sum game can be represented by an $n \times m$ payoff matrix $\mathbf{M}$ with real entries. The entry $\mathbf{M}_{ij}$ is the amount paid by the column player C to row player R, when R chooses the strategy i and C chooses the strategy j. The row player tries to maximize the payoff. If R chooses strategy i, then she is guaranteed a payoff of $\min_j M_{ij}$ and her optimal strategy will be the one that maximizes this value. Let $V_R = \max_i \min_j M_{ij}$. Similarly, the optimal strategy for C will be $V_C = \min_j \max_i M_{ij}$. When $V_R = V_C$, the game is said to have a solution(saddle point) and the value of the game is $V = V_R = V_C$.

It is also possible to have randomized or mixed strategies. A mixed strategy is a probability distribution on the set of possible strategies. The row player picks a vector $p = (p_1, \ldots, p_n)$ where p_i is the probability that R will choose strategy i. Similarly C picks $q = (q_1, \ldots, q_m)$. The payoff is now a random variable and its expected value is given by,

$$E[payoff] = p^T \mathbf{M} q = \sum_{i=1}^{n} \sum_{j=1}^{m} p_i \mathbf{M}_{ij} q_j$$

The well known Minimax Theorem of von Neumann implies that this game always has a solution:

Theorem A.1 *von Neumann's Minimax Theorem: For any two-person zero-sum game specified by matrix **M**,*

$$\max_p \min_q p^T \mathbf{M} q = \min_q \max_p p^T \mathbf{M} q$$

where min and max range over all possible distributions.

In other words, the largest expected payoff that R can guarantee by a mixed strategy is equal to the smallest expected payoff that C can guarantee. Once p is fixed $p^T\mathbf{M}q$ is a linear function of q and is minimized by setting q_j with smallest coefficient to 1. This means that if C knows the distribution p used by R, then his optimal strategy is a pure strategy. This observation leads to a simplified version of the minimax theorem:

Theorem A.2 *Loomis' Theorem: For any two-person zero-sum game specified by matrix **M**,*

$$\max_p \min_j p^T \mathbf{M} e_j = \min_q \max_i e_i^T \mathbf{M} q$$

where e_k is the unit vector with a 1 at k^{th} position.

Now consider a problem where the number of distinct inputs of a fixed size is finite, as is the number of distinct (deterministic, terminating and always correct) algorithms for that problem. We can formulate the algorithm design process as a game if we view the algorithm designer as a column player and the adversary as a row player. The pay off is some performance measure like complexity or running time. We can now restate the above theorems as follows:

Corollary A.3 *Let Π be a problem with a finite set $\mathcal{I}$ of input instances(of a fixed size) and a finite set of deterministic algorithms $\mathcal{A}$. For input $I \in \mathcal{I}$ and algorithm $A \in \mathcal{A}$, let C(I,A) denote the running time of algorithm A on input I. For probability distributions **p** over $\mathcal{I}$ and **q** over $\mathcal{A}$, let I_p denote a random input chosen according to **p** and A_q denote a random algorithm chosen according to **q**. Then,*

$$\max_p \min_q E[C(I_p, A_q)] = \min_q \max_p E[C(I_p, A_q)]$$

and

$$\max_p \min_{A \in \mathcal{A}} E[C(I_p, A_q)] = \min_q \max_{I \in \mathcal{I}} E[C(I_p, A_q)].$$

We obtain Yao's Technique from this corollary:

Proposition A.4 *Yao's Minimax Principle*

$$\min_{A \in \mathcal{A}} E[C(I_p, A_q)] \leq \max_{I \in \mathcal{I}} E[C(I_p, A_q)].$$

This means that, the expected running time of the optimal deterministic algorithm for an arbitrarily chosen input distribution is a lower bound on the expected running time of the optimal randomized algorithm. Thus, to prove a lower bound on the randomized complexity, it suffices to choose any distribution on the input and prove a lower bound on the expected running time of the deterministic algorithms for that distribution.

References

[1] S. Albers and K. Kursawe. Exploring unknown environments with obstacles. In *In Proc. 9th ACM-SIAM Sympos. Discrete Algorithms, 1998. 13*, 1998.

[2] G. J. W. Amos Fiat. *Online algorithms : the state of the art.* Berlin ; New York : Springer, 1998.

[3] M. Betke, R. L. Rivest, and M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18(2/3):231–254, 1995.

[4] S. Carlsson, H. Jonsson, and B. Nilsson. Finding the shortest watchman route in a simple polygon. In *Proc. 4th International Symposium on Algorithms and Computation, Hong Kong*, pages 58–67., 1993.

[5] W. Chin and S. Ntafos. Optimum watchman routes. *Info. Proc. Letters*, 28:39–44, 1988.

[6] W. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete and Computational Geometry*, 6(1):9–31, 1991.

[7] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment i: The rectilinear case. *Journal of the ACM*, 45:215–245, 1998.

[8] D. S. Hochbaum. *Approximation algorithms for NP-hard problems.* Boston : PWS Pub. Co., 1997.

[9] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem: A new strategy and a new analysis technique. In *Proceedings WAFR'98*, pages 211–222, Houston, 1998. A K Peters, Natick, Massachusetts.

[10] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem i : A competitive strategy. Technical Report 241, Fernuniversität Hagen, Praktische Informatik VI, 1998.

[11] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem ii : The angle hull. Technical Report 245, Fernuniversität Hagen, Praktische Informatik VI, 1998.

[12] J. O'Rourke. *Art Gallery Theorems And Algorithms.* Oxford University Press, 1987.

[13] P. R. Rajeev Motwani. *Randomized algorithms.* Cambridge University Press, 1995.

[14] X. Tan. Fast computation of shortest watchman routes in simple polygons. *Info. Proc. Letters*, 77(1):27–33, 2001.

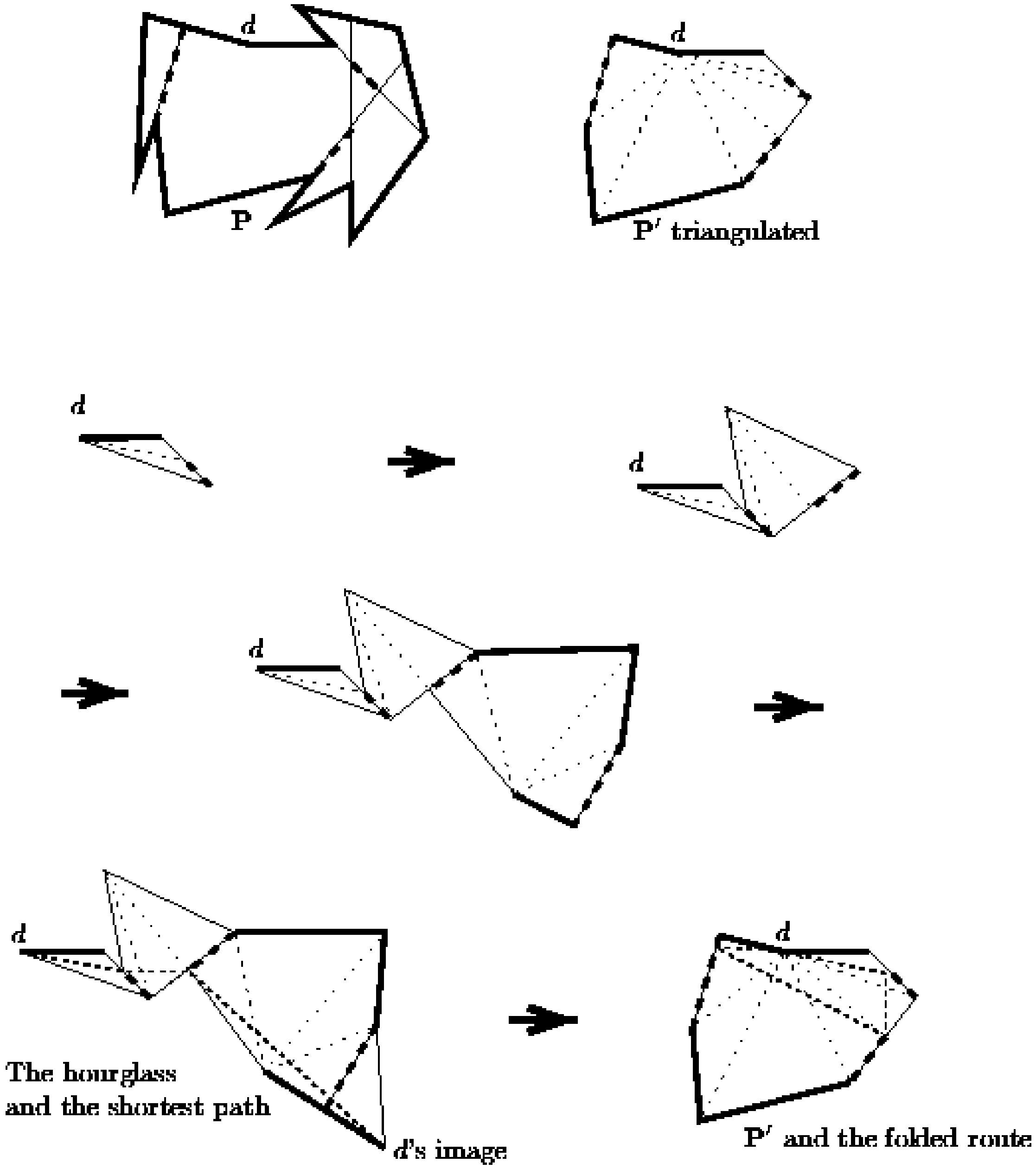

Figure 25: The unfolding procedure, the dashed lines are the active segments. (image taken from [4])