# Exploring Unknown Environments with Obstacles

Susanne Albers* Klaus Kursawe† Sven Schuierer‡

**Abstract**

We study exploration problems where a robot has to construct a complete map of an unknown environment using a path that is as short as possible.

In the first problem setting we consider, a robot has to explore $n$ rectangles. We show that no deterministic or randomized online algorithm can be better than $\Omega(\sqrt{n})$-competitive, solving an open problem by Deng, Kameda and Papadimitriou [5]. We also generalize this bound to the problem of exploring three-dimensional rectilinear polyhedra without obstacles.

In the second problem setting we study, a robot has to explore a grid graph with obstacles in a piecemeal fashion. The piecemeal constraint was defined by Betke, Rivest and Singh [3] and implies that the robot has to return a start node every so often. Betke *et al.* gave an efficient algorithm for exploring grids with rectangular obstacles. We present an efficient strategy for piecemeal exploration of grids with arbitrary obstacles.

## 1 Introduction

In robot exploration problems, a robot has to construct a complete map of an *unknown environment* using a path that is as short as possible. Many geometric and graph theoretic versions of this problem have been studied in the past [1, 2, 3, 4, 5, 6, 8, 9, 10, 11]. A general problem setting was introduced by Deng, Kameda and Papadimitriou [5]. The robot is placed in a room with obstacles. The exterior wall of the room as well as the obstacles are modeled by simple polygons. Figure 1 shows an example in which the room is a rectangle and all obstacles are rectilinear. The robot has 360° vision. Its task is to move through the scene so that it sees all parts of the room. More precisely, every point in the room must be visible from some point on the path traversed. The problem is interesting for several polygon types (general, rectilinear, convex, rectangular). Even the case that no obstacles are placed in the room has been the subject of extensive studies [5, 8, 9, 11].

Given a scene $S$, let $L_A(S)$ be the length of the path traversed by algorithm $A$ to explore $S$. Since $A$ does not know $S$ in advance it is also referred to as an *online algorithm.* Let $L_{OPT}(S)$ be the length of the path of an optimum algorithm that *knows the scene in advance.* Following Sleator and Tarjan [13] we call an online exploration algorithm $A$ *c-competitive* if for all scenes $S$, $L_A(S) \le c \cdot L_{OPT}(S)$. Here we assume that $A$ is a deterministic algorithm.

In the scenario above it is assumed that the robot can see an infinite range as long as no obstacle or exterior wall blocks the view. However, in practice, a robot's sensors can often scan only a distance of a few meters. This situation can be modeled by adding a grid to the scene, as shown in Figure 2, and requiring that the robot moves on the nodes and vertices of the grid. A node in the grid models

*Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. Email: albers@mpi-sb.mpg.de

†Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. Email: kursawe@mpi-sb.mpg.de

‡Institut für Informatik, Am Flughafen 17, Geb. 051, D-79110 Freiburg, Germany. Email: schuiere@informatik.uni-freiburg.de

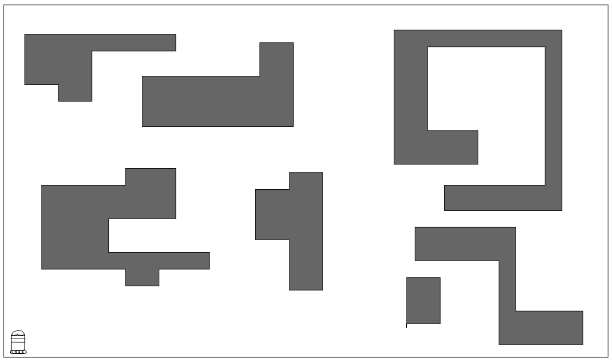

Figure 1: A sample scene

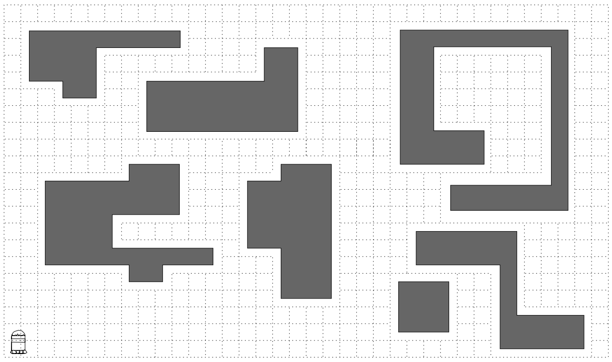

Figure 2: A sample scene with a grid

the vicinity that the robot can see at a given point. Now the robot has to explore all nodes and edges of the grid using as few edge traversals as possible. A node is explored when it is *visited* for the first time and an edge is explored when it is *traversed* for the first time. At any node the robot knows its global position and the directions of the incident edges. Betke, Rivest and Singh [3] introduced an interesting, more complicated variant of this problem where an additional *piecemeal constraint* has to be satisfied, i.e, the robot has to return to a start node $s$ every so often. These returns might be necessary because the robot has to refuel or drop samples collected on a trip.

**Previous results:** In [5], Deng, Kameda and Papadimitriou first studied the problem of exploring rooms without obstacles. They give a deterministic 2-competitive online algorithm for exploring rectilinear rooms if the length of the path is measured in the $L_1$-metric. The start and end point of the exploration path have to be the same; otherwise the competitive ratio is slightly higher. Kleinberg [11] shows than no deterministic online exploration algorithm can be better than $\frac{5}{4}$-competitive (again in the $L_1$-metric). Recently, Hoffmann, Icking, Klein and Kriegel [8, 9] developed deterministic exploration algorithms for general rooms, i.e., the exterior wall can be an arbitrary polygon. The competitiveness achieved by their best algorithm is 26.5.

Deng, Kameda and Papadimitriou [5] also give lower and upper bounds on the competitive ratio for exploring rooms with obstacles. They show that no algorithm for rooms with obstacles can achieve a constant competitive ratio. The obstacles needed in the construction are long, thin diamonds. Thus, the lower bound does not necessarily hold for restricted classes of obstacles such as rectangles or rectilinear objects. Deng, Kameda and Papadimitriou also develop an $O(n)$-competitive algorithm for exploring rectilinear rooms with $n$ rectilinear obstacles. They conjectured that there exists an algorithm that achieves a constant competitive ratio.

Kalyanasundaram and Pruhs consider the problem of exploring a scene that contains $n$ convex obstacles [10]. However, their model is different in that they only require that all the edges of the obstacles have to be seen. At first glance, this problem seems to be equivalent to the problem of exploring the whole environment, but Figure 3 illustrates a difference: An offline algorithm does not have to explore the entire scene. Kalyanasundaram and Pruhs show that in this model $\Omega(\min(n, \sqrt{n\alpha}))$ is a lower bound on the competitive ratio achieved by any online algorithms, where $\alpha$ is the *aspect ratio* of the scene. The aspect ratio of an obstacle $O$ is defined as $R/r$, where $R$ is the radius of the smallest circle that circumscribes $O$ and $r$ is the largest circle that inscribes $O$. The aspect ratio of a scene is the maximum of the aspect ratios of the obstacles. The scene of Figure 3 already gives a lower bound of $\Omega(\sqrt{n})$, where $n$ is the number of rectangles. Kalyanasundaram and Pruhs also present a strategy with a competitive ratio that matches the lower bound of $\Omega(\min(n, \sqrt{n\alpha}))$ up to a constant factor. We note that the lower bound does not hold for the model studied by Deng, Kameda and

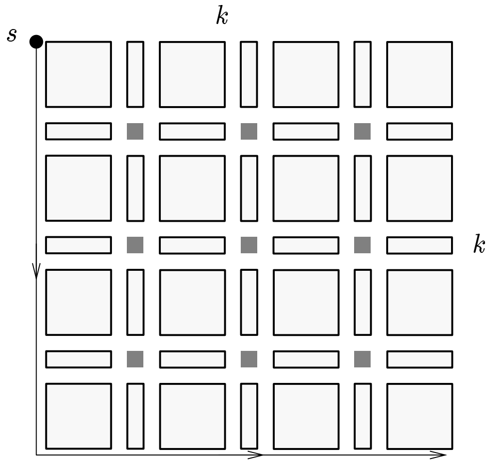


Figure 3: If an offline strategy needs to see only the boundary of all obstacles, then it can follow the path shown in the figure. By making the gaps between the rectangles small enough, this path does not see the $O(k^2)$ darkly shaded areas within the scene. An online algorithm has to visit all these areas since it has to make sure that no obstacle is contained in one of them.

Papadimitriou that we will consider in this paper.

Betke, Rivest and Singh [3] developed two algorithms for piecemeal exploration of grids with rectangular obstacles. The algorithms, called *Wavefront* and *Ray*, need $O(|E|)$ edge traversals where $|E|$ is the number of edges in the graph. This is optimal. The *Wavefront* algorithm implements a breadth-first strategy while the *Ray* algorithm implements a simple and elegant depth-first strategy.

**Our contribution:** In Section 2 of this paper we first present lower bounds for the problem of exploring a room with $n$ rectangles. We show that no deterministic or randomized online exploration problem can be better than $\Omega(\sqrt{n})$-competitive, disproving the conjecture by Deng, Kameda and Papadimitriou. Our proof is based on a new recursive construction of a scene. The robot repeatedly has to find relevant parts of the room in which obstacles are located. We can extend our bound to three-dimensional scenes without obstacles. We show that no algorithm for exploring the interior of a rectilinear polyhedron with $n$ vertices can be better than $\Omega(\sqrt{n})$-competitive.

In Section 3 we study piecemeal exploration. We present an algorithm that explores a grid with arbitrary obstacles using $O(|E|)$ edge traversals, which is optimal. Our algorithm is a generalization of the *Ray* algorithm by Betke, Rivest and Singh. In the original *Ray* algorithm for rectangular obstacles, it is required that the robot always knows a path back to the start node whose length is most the radius of the graph. When exploring grids with arbitrary obstacles, this constraint cannot be satisfied. We solve this problem by presenting an efficient strategy for exploring the boundary of arbitrary obstacles.

## 2 Lower bounds

In this section we give lower bounds on the competitive ratios achieved by deterministic and randomized online exploration algorithms in two-dimensional scenes.

### 2.1 A lower bound for deterministic online algorithms

**Theorem 1** *Let $A$ be a deterministic online algorithm for exploring two-dimensional scenes with $n$ rectangles. If $A$ is $c$-competitive, then $c = \Omega(\sqrt{n})$.*

In the remainder of this section we prove this theorem. Given an online exploration algorithm $A$, we construct a scene with $n$ rectangles such that the path used by $A$ is at least $\Omega(\sqrt{n})$ times as long as the path used by an optimal offline algorithm OPT. We show how to construct the scene and then analyze the paths by $A$ and OPT.

**The construction:** Let $n$ be a positive integer and $k = \lfloor\sqrt{n/2}\rfloor$. Let $\varepsilon = \frac{1}{2}(2k)^{-k}$. The obstacles used in the construction are *$k$-combs*, as depicted in Figure 4.

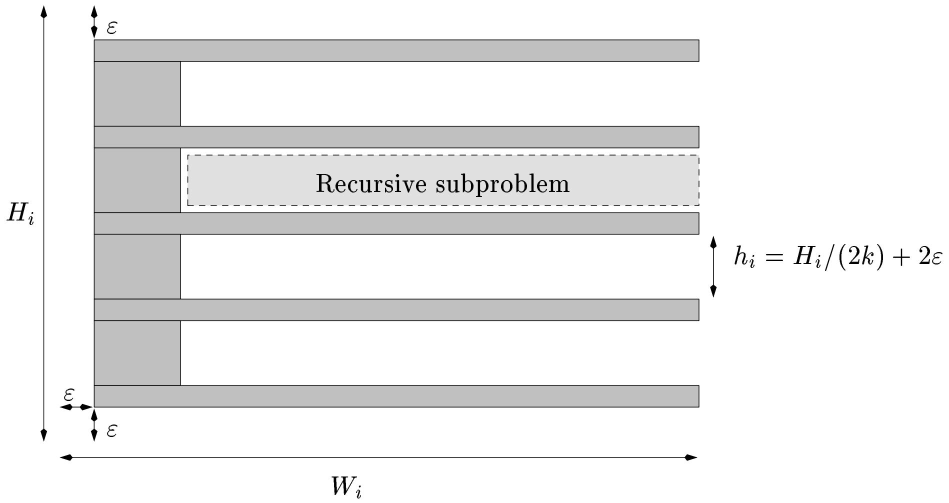


Figure 4: A $k$-comb in stage $i$.

A $k$-comb consists of $k$ *spike rectancles*, or *spikes* for short, that span the whole width of the $k$-comb, and $k-1$ *base rectangles* that have width $w_b = 1$. The side of a base rectangle that is aligned with the spikes is called the *outer side* of the base rectangle; the opposite side is called the *inner side*. The distance between a spike and a base rectangle is $\varepsilon' = \varepsilon^2/(2k)$. This is illustrated in Figure 5.

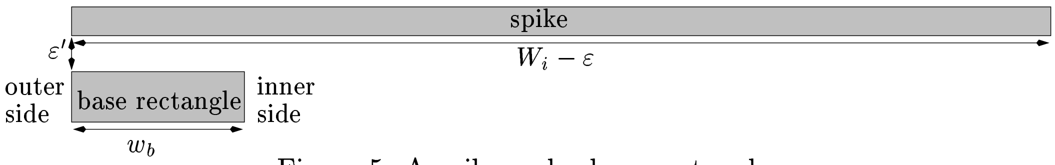


Figure 5: A spike and a base rectangle.

We use a recursive construction where the rectangles become very thin. We show later how to modify the scene so that a unit diameter circle can be inscribed into any rectangle.

The construction of the scene proceeds in $k$ *stages*. In each stage $i$, $1 \le i \le k$, exactly one $k$-comb is used. Thus, $k(2k-1) \in \Theta(n)$ rectangles are placed in total. The $k$-comb of the first stage has height $k$ and width $2k$. The crucial property of our construction is that the $k$-comb of stage $i$, $2 \le i \le k$, is placed between two spikes of stage $i-1$.

For $i = 1, \dots, k$, let $W_i$ be the length available in $x$-direction to place the $k$-comb of stage $i$ and let $H_i$ be the length available in $y$-direction (see Figure 4). We set $H_1 = k$ and $W_1 = 2k$. In every Stage $i$, $2 \le i \le k$, the distance between the current $k$-comb and the rectangles that belong to the $k$-comb of Stage $i-1$ is $\varepsilon$. For the width $W_i$, $1 \le i \le k$, we obtain

$$W_i = 2k - (w_b + \varepsilon)(i-1) \ge k. \tag{1}$$

We set the height $h_i$ of a base rectangle in stage $i$ to $(H_i/2k) + 2\varepsilon$. The height of a spike is then given by $(H_i' - 2\varepsilon)/k$ where $H_i' = H_i - (k-1)(H_i/2k + 2(\varepsilon + \varepsilon'))$. This implies that $H_{i+1} = H_i/(2k)$. We conclude

$$H_i = H_1/(2k)^{i-1}. \tag{2}$$

We now specify the placement of the $k$-comb for stage $i+1$ in stage $i$, $1 \le i \le k$. Recall that the exploration algorithm $A$ used by the robot is deterministic. Assume that the robot is located at the start point of the scene, which is one of the four corners of the $k$-comb in stage $i$. The $k$-comb of stage $i$ is placed such that the robot faces the outer sides of the base rectangles. Thus, if the robot enters stage $i$ to the right of a spike in stage $i-1$, then the $k$-comb is a mirror image of the arrangement given in Figure 4. To find the $k$-comb of stage $i+1$, the robot has to explore the inner sides of the base rectangles. It will succeed only at the very last base rectangle. Alternatively, the robot can move to the other side of the $k$-comb. In this case the $k$-comb of stage $i+1$ may be between two spikes of any unexplored base rectangle.

**The analysis:** We first study OPT. To explore stage $i$ of the construction, OPT can move as follows. We assume that the robot is located in the lower left corner of the stage.

1. Move a distance of $H_i$ upwards, along the left side of the obstacles.
2. If $i < k$, move to the lower left corner of the $k$-comb for stage $i+1$ and explore stage $i+1$ recursively. Otherwise, if $i = k$, move to the right of the $k$-comb.
3. Move upwards and then again downwards along the right side of the $k$-comb.

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

$$L^i_{OPT} \le 2H_i + w_b + \varepsilon + 2H_i = 4H_i + w_b + \varepsilon.$$

If $i = k$, then the distance traveled is bounded by $4H_k + 2k - (k-1)(w_b + \varepsilon)$ where the additive factor of $2k - (k-1)(w_b + \varepsilon)$ accounts for travel to the right side of the $k$-comb. Summing up and using (2), we obtain that the length of the path used by OPT is bounded by

$$L_{OPT} \le \sum_{i=1}^{k-1}(4H_i + w_b + \varepsilon) + 4H_k + 2k - (k-1)(w_b + \varepsilon) = \sum_{i=1}^{k}(4H_1)/(2k)^{i-1} + 2k \le 10k. \tag{3}$$

For the analysis of the online algorithm $A$ we only consider the distance traveled in $x$-direction. When the robot is located at a lower corner of stage $i$, it has two possibilities to find the $k$-comb of the next stage. It can (a) explore the inner sides of all base rectangles or (b) change sides (that is, it moves from the left to the right side of the $k$-comb, or vice-versa). In the first case the $k$-comb of the next stage, which is a distance of $\varepsilon$ below the upper boundary of the base rectangle (and above the lower boundary, respectively), only becomes visible at a distance of at most $\varepsilon$ to the inner side of the base rectangle since the distance between a base rectangle and a spike is $\varepsilon' = \varepsilon^2/2k$. This is illustrated in Figure 6.

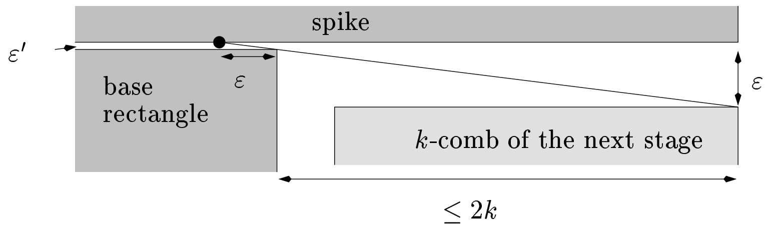


Figure 6: The $k$-comb of the next stage only becomes visible at a distance of $\varepsilon$ to the inner side of a base rectangle.

Hence, the robot has to travel at least a distance of $2(w_b - \varepsilon)$ for each of the $k-1$ base rectangles. In the second case the robot travels at least a distance of $W_i \geq k$. Thus, the length of the path traveled by $A$ is at least $\sum_{i=1}^{k} k = k^2$ and Theorem 1 follows because by (3) we have $L_{OPT} \in O(k)$.

**The size of the obstacles:** Our construction needs very small rectangles. The rectangles in stage $k$ have a height of $h$ with $\frac{1}{2}(2k)^{-k} < h < k^{-k}$. This problem can be solved by multiplying all lengths by a factor of $2(2k)^k$. Suppose that all lengths are multiplied by a factor of $x$. It is easy to see that the length of the path used by OPT increases by a factor of exactly $x$. In the same way, the length of the path used by $A$ increases by a factor of $x$ and the competitive ratio remains the same.

### 2.2 A lower bound for randomized online algorithms

**Theorem 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 randomize the construction given in the previous section and show that the expected length of the path used by any deterministic online exploration algorithm $A$ is at least $\Omega(\sqrt{n})$ times the length of the path used by OPT. The theorem then follows from Yao's minimax principle [14]. In each stage, the $k$-comb is placed so that with probability 1/2 the outer sides of the base rectangles are to the left side, and with probability 1/2 they are to the right side. In each stage the position of the $k$-comb of the next stage is chosen randomly. Each of the $k-1$ configurations occurs with equal probability.

The upper bound on the length of the path used by OPT does not change. Consider any deterministic online algorithm $A$. When entering stage $i$, with probability 1/2, the robot is located on the opposite side as the base rectangles and, thus, incurs no cost in $x$-direction. With probability 1/2, the robot is located on the same side as the base rectangles. The robot can change sides, traversing a path of length at least $k$. If the robot stays on the same side, the expected number of base rectangles it has to explore before finding the $k$-comb of the next stage is $k/2$. Thus, the total length of the path traversed by $A$ reduces by a factor of 4 but it is still in $\Omega(k^2)$. □

### 2.3 Three-dimensional scenes

**Theorem 3** *Let $A$ be an online algorithm for exploring a simple three-dimensional rectilinear polyheron without obstacles. If $A$ is $c$-competitive, then $c = \Omega(\sqrt{n})$ where $n$ is the number of vertices of the polyhedron.*

This result was independently obtained by Frank Hoffmann [7] based on a preliminary version of this paper that contained the lower bounds for the two-dimensional case.

**Proof**: To explore a three-dimensional polyhedron, the robot is allowed to move not only in $x$- and $y$-direction but also in $z$-direction. We construct a polyhedron such that the rectangles are no longer independent objects in the scene but part of the hull of the polyhedron. We first modify the above construction in the following way. We place a rectangle $R$ around the scene constructed above and we join each base rectangle with the spike above it. In this way we obtain $k-1$ $L$-shaped obstacles in each $k$-comb. Let $P$ be the polygon enclosed by $R$ where the holes of $P$ are given by the $L$-shaped obstacles and the remaining spikes.

We now extend each of the obstacles used in stage $i$, $1 \leq i \leq k$ a distance of $2(k+1-i)$ in the $z$-direction. The rectangle $R$ is extended a distance of $2(k+1)$ in the $z$-direction. All $L$-shaped obstacles are closed from the top by a cover. The cover is a cuboid of height 1 that just fits the

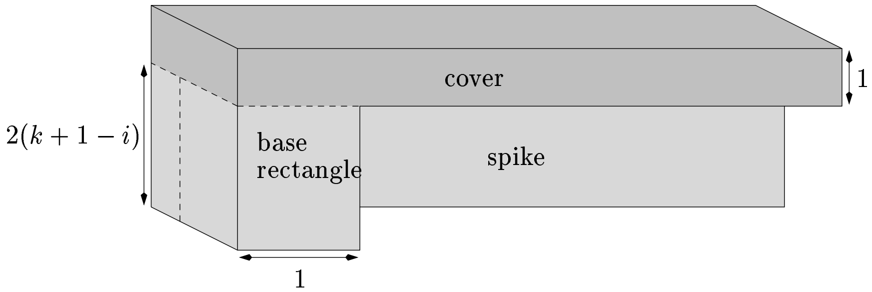


Figure 7: An $L$-shaped obstacle in stage $i$ with a cover.

$L$-obstacle (see Figure 7). The remaining spikes of the $k$-combs also get a cover, that is, they are extended one unit in $z$-direction. The distance to the cover of stage $i-1$ is 1. We close the lower side of our construction at $z = 0$ with the polygon $P$ and the upper side at $z = 2(k+1)$ with the rectangle $R$. The surfaces we have constructed now define a simply connected rectilinear polyhedron. In the beginning the robot is located in the lower left corner of the polyhedron at height $z = 2(k+1)$.

In each stage the robot now has three options. First it can travel above the covers and "peek" below each cover at a cost of $2(1-\varepsilon)$. Or it can stay on the left side and explore the inner sides of the $L$-shaped obstacles. Or it can change sides. It may happen that the robot moves in each stage some distance from left to right by traveling above the covers. When it finally changes sides, the distance to the right side can be very small. In this case, we accumulate the distance traveled from left to right in the previous stages and charge it to the current stage. Thus, in each case the distance traveled by the robot is again at least $k$. Note that the robot cannot pass below the obstacles.

To explore the polyhedron, OPT moves two units down, right below the covers, and executes the algorithm for the two-dimensional case. In general, when entering stage $i$, OPT is above the covers of that stage. It moves two units downwards, explores the left sides of the obstacles and enters the next stage. After having reached stage $k$, the robot changes sides and moves two units up when exiting a stage. The path traversed by OPT increases only by an additive factor of $O(k)$. $\square$

# 3 Exploring grids with arbitrary obstacles

Consider a grid graph with arbitrary obstacles and let $s$ be the start node the robot has to relocate to. Let $r$ be the radius of the graph, i.e., $r$ is the maximum of all shortest path distances between $s$ and any other node in the graph. We assume that the robot can traverse a total of $2(3+\alpha)r$ edges, for some constant $\alpha > 0$, between two consecutive visits to $s$. Thus, it can traverse $R = (3+\alpha)r$ edges before moving back to $s$. We present an algorithm that explores an unknown grid with arbitrary obstacles using $O(|E|)$ edge traversals.

## 3.1 The algorithm

The algorithm we develop is a generalization of the *Ray* algorithm proposed by Betke, Rivest and Singh [3]. The original *Ray* algorithm only explores grids with rectangular obstacles. It is essential in the algorithm that the robot always knows a path back to $s$ that has a length of at most $r$. When exploring grids with arbitrary obstacles we cannot always satisfy such a constraint. We solve this problem by presenting an efficient strategy for exploring the boundary of arbitrary obstacles.

We assume that the exterior boundary of the grid is a rectangle and that no obstacle touches the exterior boundary. Moreover, for simplicity, we assume that (1) the start node $s$ is located in the

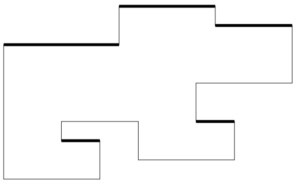

Figure 8: The open segments

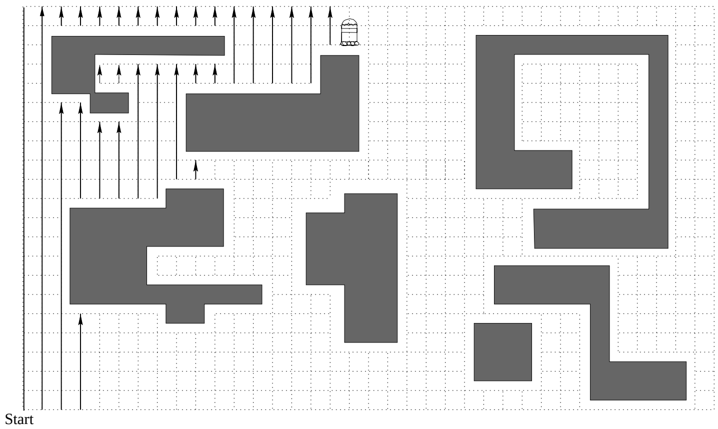


Figure 9: The *Ray* algorithm

bottom left corner of the scene and (2) that no edge belongs to the boundary of two obstacles. In the appendix we show how to handle the general case. At any time *dist* denotes the number of edges traversed by the robot since the last visit to $s$.

A node or an edge on the boundary of an obstacle is *explored* when it is visited or traversed, respectively, during an excecution of the procedure *Map-Obstacles* that we will decribe in detail below. All other nodes and edges in the grid are explored when they are visited or traversed, respectively, for the first time. A node or edge that is not explored is also called *new*.

In a first step, our *Ray* algorithm explores the exterior boundary of the scene. Starting from $s$, it travels in clockwise direction along the boundary until $s$ is reached again. Knowing the exterior boundary will be convenient because the robot can then distinguish it from the unexplored boundary of obstacles. During the exploration of the exterior boundary, whenever $dist = R$ at some node $x$, the robot executes a procedure *Refuel* that we will describe later. In this procedure the robot moves back to $s$ and then relocates again to $x$ to resume exploration.

An obstacle is *new* if none of the nodes and edges on the boundary are explored. An obstacle is *mapped fully* if all nodes and edges on the boundary are explored. For an obstacle $O$, an *open segment* $S$ of $O$ is a maximal sequence consecutive horizontal edges on the boundary of $O$ such that the interior of $O$ is to the south of $S$. In Figure 8 the open segments are shown in bold lines.

The actual exploration of the scene proceeds in rays. Let $S$ be the bottom segment of the exterior boundary and let $s = x_1, \ldots, x_n$ be the vertices of $S$. For every $i$, $1 \leq i \leq n$, the robot starts at node $x_i$ and explores a vertical ray of edges in northern direction until an obstacle or the exterior boundary is hit. Then the robot backtracks to $x_i$ and moves to the neighboring node $x_{i+1}$ if it exists. In general, we say that the robot *hits an obstacle* if it reaches a node on the boundary with degree less than 4. We refine this definition when rays are traversed. Here the robot *hits an obstacle* if the obstacle blocks the ray, i.e. there is no outgoing edge in northern direction.

An important feature of the *Ray* algorithm is that it applies a depth-first strategy, as illustrated in Figure 9. Whenever the robot hits a new obstacle $O$ at a node $y$ while exploring a ray $R_i$ started at $x_i$, the robot immediately explores the boundary of $O$ using a procedure *Map-Obstacles* that we will explain below. When *Map-Obstacles* terminates, the robot is again located on $y$. The robot then visits the open segments of $O$ in clockwise direction and explores vertical rays started at these segments. The same strategy is applied recursively to new obstacles that are discovered while the robot explores rays started at $O$'s open segments. When all the open segments of $O$ are visited, the robot moves to $y$ and backtracks along $R_i$ to $x_i$. While exploring vertical rays, at any node the robot also moves one edge to the west to explore horizontal edges.

A formal description of the exploration along rays is given in the procedure *Shoot-Rays*. Given a segments $S$, which is either the bottom segment of the exterior boundary or an open segment of an obstacle, the robot explores vertical rays starting at the nodes of $S$. We require that at the beginning of the procedure, the robot is located at one of the endnodes of $S$.

**Algorithm Ray**

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

**Procedure Shoot-Rays($S$)**

1. Let $x$ be the current node where the robot is located and let $x = x_1, \ldots, x_k$ be the 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 exterior boundary or some obstacle is hit. Execute *Refuel* whenever $dist = R$;
5. $y :=$ current node;
6. **if** a new obstacle was hit **then**
7. *Map-Obstacles*;
8. Let $S_1, \ldots, S_l$ be the open segments of $O$ in clockwise order w.r.t. $y$;
9. **for** $j := 1$ **to** $l$ **do**
10. Move in clockwise direction along $O$ to first endpoint of $S_j$. Execute *Refuel* whenever $dist = R$;
11. *Shoot-Rays*($S_j$);
12. Move to $y$ along the boundary of $O$. Execute *Refuel* whenever $dist = R$;
13. Move to $x_i$ following the ray $R_i$ backwards. Execute *Refuel* whenever $dist = R$;

In *Shoot-Rays*, when exploring vertical rays or moving around obstacles to reach open segments, the robot executes *Refuel* whenever $dist = R$. Let $x$ be the node the robot is located on when $dist = R$. In *Refuel* the robot first moves back to $s$. In a command "Move back to $s$" the robot follows the path that is has traversed since the last visit to $s$. Then the robot relocates to $x$: It computes a shortest path $P$ from $s$ to $x$ assuming all obstacles are known. If the robot hits a new obstacle $O$ while traversing $P$, it explores $O$ using *Map-Obstacles* and then computes a new shortest path from the current node to $x$. Every time $dist = R$, the robot moves back to $s$.

**Procedure Refuel**

1. $x :=$ current node where the robot is located;
2. Move back to $s$;
3. **while** $x$ is not reached **do**
4. $P :=$ shortest path from current node to $x$ assuming that all obstacles are known;
5. Move along $P$ until some new obstacle is hit or $dist = R$ or $x$ is reached;
6. **if** some new obstacle was hit **then**
7. *Map-Obstacles*;
8. **if** $dist = R$ **then**
9. Move back to $s$;

The crucial part of our *Ray* algorithm is the procedure *Map-Obstacles* that is called every time the robot hits a new obstacle $O$ at a node $x$. When *Map-Obstacles* terminates, $O$ and all obstacles hit during the execution of the procedure are mapped fully and the robot is located again on $x$,

Initially, the robot moves in, say, clockwise direction along the boundary of $O$ and tries to reach $x$ again while $dist \leq R$. If it succeeds in reaching $x$, then $O$ is mapped fully and the call of *Map-Obstacles* terminates. If the robot cannot reach $x$ while $dist \leq R$, then the call is more involved. An obstacle is *mapped partially* if some nodes or edges on the boundary are explored but the obstacle is not mapped fully. Note that boundary edges that have been traversed only when the robot moved along rays in *Shoot-Rays* are still considered unexplored. Given a partially mapped obstacle, we call a maximal sequence of explored nodes and edges on the boundary an *explored segment*. An explored segment may consist of only one visited node.

During the execution of *Map-Obstacles*, we maintain a set of partially mapped obstacles. More precisely, we maintain a set $B$ of *breakpoints*, where each breakpoint $b \in B$ is the endpoint of an explored segment. Initially, $x$ is inserted into $B$. If the robot cannot move around $O$ while $dist \leq R$, then the node $z$ reached when $dist = R$ is added to $B$. The robot moves back to $s$ and computes a shortest path $P$ from $s$ to $z$ assuming all obstacles are known. With respect to partially mapped obstacles we require that $P$ does not go through interior nodes of explored segments.

The exploration then proceeds in *phases* until $B = \emptyset$. In each phase the robot makes progress towards exploring the boundary of partially explored obstacles. At the beginning of a phase the robot is given a path $P$ from the current node to the most recently inserted breakpoint $b_0 \in B$. The robot then travels along $P$. We distinguish three cases.

**Case 1:** The robot hits $b_0$ or some other $b \in B$.
In this case $b_0$ or $b$ is deleted from $B$ provided that the breakpoint is incident to an explored boundary edge, i.e. $b_0$ or $b$ is not an isolated explored boundary node that is incident to only unexplored boundary edges. The robot then moves along the unexplored boundary of the partially mapped obstacle until $dist = R$ or some other breakpoint $b' \in B$ is reached, see Figure 10. If some $b' \in B$ is reached, $b'$ is deleted from $B$ if no unexplored boundary starts at $b'$. Otherwise, if $dist = R$, the current node is inserted as breakpoint into $B$. In any case the robot moves back to $s$ and computes a shortest path $P$ from $s$ to the most recently inserted breakpoint in $B$.

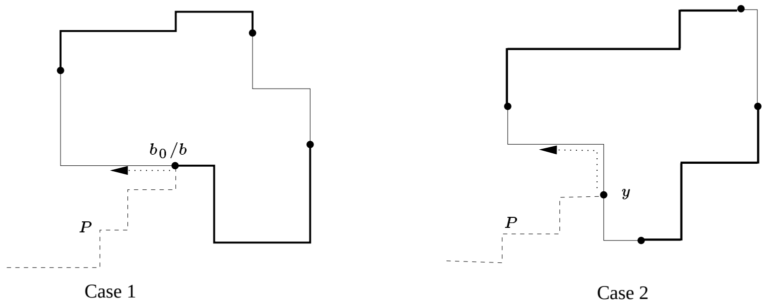


Figure 10: Exploration of partially mapped obstacles

**Case 2:** The robot hits a node $y$ that belongs to the unexplored boundary of an obstacle.
Node $y$ is inserted into $B$ and a Boolean variable *new* is set to true, indicating that a new breakpoint was inserted. The robot moves along the unexplored boundary until some $b \in B$ is reached or $dist = R$, see Figure 10. If $dist = R$ and no breakpoint was reached, the current node is inserted into $B$ and the robot moves back to $s$. If some $b \in B$ was reached and $b = y$, then the robot successfully mapped a new obstacle. Node $y = b$ is deleted from $B$ if no unexplored boundary starts at $b$. The robot then computes a new shortest path $P$ from $y$ to $b_0$. If the robot reaches some $b \in B$, $b \neq y$, 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 $y$ (if it does not run out of fuel) and explore new boundary edges starting from $y$. In any case, *new* is set to false.

**Case 3:** The robot has not reached a $b \in B$ or a node $y$ on unexplored boundary but $dist = R$. The robot moves back to $s$ and computes a new shortest path from $s$ to $b_0$.

In the end, if $B = \emptyset$ but the robot is not located on $x$, we insert $x$ into $B$ (in this special case $x$ is not the endpoint of an explored segment). The robots computes again a shortest path from the current node to $x$. If the path hits new obstacles, the same exploration in phases starts again. In the pseudocode below a phase starts in line 22 and ends in the following execution of line 21. We have chosen this presentation so as to incorporate the initial movement of the robot around the new obstacle $O$ hit at $x$.

**Procedure Map-Obstacles**

1. $x :=$ current node robot is located; $B := \{x\}$; $new :=$ true; $y := x$;
2. **while** $B \neq \emptyset$ **do**
3.     **if** $dist < R$ **then**
4.         Move along unexplored boundary until some $b \in B$ is reached or $dist = R$;
5.         **if** some $b \in B$ was 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** $B = \emptyset$ and current node $\neq x$ **then**
16.         $B := \{x\}$;
17.     **if** $B \neq \emptyset$ **then**
18.         **if** $new =$ true **then**
19.             $P :=$ path from current node to most recently inserted node in $B$ that follows the boundary just explored; $new :=$ false;
20.         **else**
21.             $P :=$ shortest path from current node to most recently inserted node in $B$ assuming all obstacles are known;
22.         Move along $P$ until some $b \in B$ is reached or some unexplored boundary is hit at a node $y$ or $dist = R$;
23.         **if** some $b \in B$ was reached and $b$ is incident to an explored boundary edge **then**
24.             Delete $b$ from $B$;
25.         **else if** some unknown boundary was hit at $y$ **then**
26.             Insert $y$ into $B$; $new :=$ true;

## 3.2 The analysis of the *Ray* algorithm

### 3.2.1 Correctness

We first show that the procedure *Map-Obstacles* works correctly.

**Lemma 1** *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.*

*a) The obstacles hit at the beginning and during the execution are mapped fully.*

*b) The robot is located at the node where the execution started.*

**Proof**: We first show that during an execution of *Map-Obstacles*, for any partially mapped obstacle, the two endpoints of any explored segment are breakpoints in $B$. At the beginning of the execution the current node $x$ of the new obstacle $O$ is inserted into $B$. The robot then moves along the unexplored boundary of $O$. In general, whenever the robot hits a node $y$ on unexplored boundary, $y$ is inserted into $B$ in line 26 and the robot moves along the unexplored boundary in the following execution of line 4. Whenever the robot has to interrupt the exploration of boundary edges because $dist = R$, the current node is inserted into $B$ in line 13. In line 10 a breakpoint $b$ is only deleted from $B$ if no unexplored boundary starts at $b$. If a breakpoint is deleted in line 24, the robot moves along the unexplored boundary in the following execution of line 4 and a breakpoint is inserted in line 13 when $dist = R$.

Part a) now follows from the above statement and the fact that $B = \emptyset$ when *Map-Obstacles* terminates. For the proof of part b) we observe that if $B = \emptyset$ and the current node is not equal to the initial node $x$, then $x$ is inserted into $B$ in line 16. Thus, when *Map-Obstacles* terminates, $B = \emptyset$ and the robot is located on the node where the execution started. $\Box$

By the above lemma, an execution of *Map-Obstacles* starts and ends at the same node. Thus the moves of the robot after lines 8 in *Shoot-Rays* and *Refuel* are well-defined. We conclude that all moves during the *Ray* algorithm are well-defined. Every obstacle is definitely hit when the robot traverses vertical rays in *Shoot-Rays*. Thus all nodes and edges on the boundary of obstacles are explored. Any other vertical edge in the grid lies on a vertical ray started from the bottom segment of the exterior boundary or from an open segment of some obstacle, and thus is being explored. Any other horizontal edge in the grid is explored when the robot moves along vertical rays. Thus, when the *Ray* algorithm terminates, the entire scene is explored.

### 3.2.2 Counting the number of edge traversals

In the following, given a path $P$, let $|P|$ denote the length of $P$, i.e. $|P|$ is the number of edges of $P$.

The robot is initially located on $s$. Let $N$ be the total number of visits to $s$ before the robot has explored the entire scene. For $i = 1, \ldots, N$, 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 $s$. We will show that $\sum_{i=1}^{N} |Q_i| \in O(|E|)$. Since the total number of edge traversals is bounded by $2\sum_{i=1}^{N} |Q_i|$, the desired result follows.

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

Our first goal is to show that if a path $Q_i$, $1 \leq i \leq N$, traverses $R = (3 + \alpha)r$ edges, then it traverses at least $\alpha r/2$ fresh edges. We begin with some basic lemmas.

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

**Proof**: Initially, in line 4 of *Refuel*, the robot computes a shortest path $P$ from $y$ to $x$ assuming that all obstacles are known. Thus $|P| \leq 2r$. Since the robot has traversed $2r+l$ edges, it must have hit an obstacle. First assume that the robot is currently exploring the boundary of an obstacle $O$ in line 7 but has not hit other new obstacles so far. When hitting $O$, less than $|P|$ edges were traversed. All edges traversed on the boundary of $O$ are fresh. Thus the number of fresh edges traversed is at least $2r + l - |P| \geq l$.

We study the case that the robot has hit $k$, $k \geq 1$, new obstacles so far and was able to move around them in line 7. Let $y_j$ be the node where $O_j$ is hit, $1 \leq j \leq k$. Every time the robot manages to move around an $O_j$ and reach $y_j$ again, it computes a shortest path from $y_j$ to $x$ in the following execution of line 4. Let $P_j$ be this path and let $P'_j$ be the path traveled by the robot between $y$ and the first visit to $y_j$ (see Figure 11). By $|O_j|$ we denote the number of boundary edges on $O_j$. Let $P'_0$

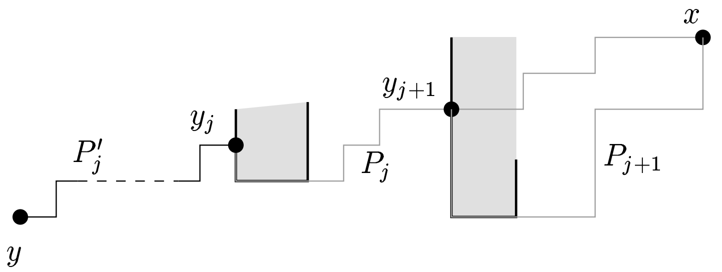


Figure 11: The definition of $P_j$ and $P'_j$.

be the empty path and $P_0 = P$. We prove inductively that $|P'_j| + |P_j| \leq |P| + 2\sum_{i<j} |O_i| + |O_j|$ holds for $j = 0, \ldots, k$. Here we set $|O_0| = 0$. For $j = 0$ there is nothing to show. Assume that the inequality holds for $j \geq 0$. Let $P(y_j, y_{j+1})$ be the path traveled by the robot on $P_j$ between $y_j$ and the first visit to $y_{j+1}$. We have $|P'_{j+1}| = |P'_j| + |O_j| + |P(y_j, y_{j+1})|$ and $|P_{j+1}| \leq |P_j| - |P(y_j, y_{j+1})| + |O_{j+1}|$. Thus $|P'_{j+1}| + |P_{j+1}| \leq |P'_j| + |P_j| + |O_j| + |O_{j+1}| \leq |P| + 2\sum_{i<j+1} |O_i| + |O_{j+1}|$.

We conclude 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, i.e., $l \leq 2\sum_{j=1}^{k} |O_j|$. Since all edges traversed when exploring $O_1, \ldots, O_k$ are fresh, the desired bound follows.

Finally assume that the robot is currently moving around a new obstacle $O_{k+1}$ and has already hit $k$ new obstacles $O_1, \ldots, O_k$. If the robot has traversed at most $2r$ edges when reaching $O_{k+1}$, then the desired bound follows because all edges traversed on $O_{k+1}$ are fresh. Suppose that the robot has traversed $2r + l'$, $l' > 0$, edges when reaching $O_{k+1}$. The robot has traversed at most $2r + l' \leq |P'_k| + |P_k| \leq |P| + 2\sum_{j=1}^{k} |O_j| \leq 2r + 2\sum_{j=1}^{k} |O_j|$ edges, at least $\sum_{j=1}^{k} |O_j|$ of which were fresh. Thus at least $l'/2$ fresh edges were fresh. When traversing the boundary of $O_{k+1}$ only fresh edges are traversed and the total number of fresh edges on the path is $l - l' + l'/2 \geq l/2$. $\Box$

We want to show a similar statement for *Map-Obstacles*. We need one more definition and a useful proposition. If during an execution of *Map-Obstacles* the robot hits a partially mapped obstacle at a node $y$, let $y_l$ be the nearest explored node on the boundary of $O$ if the robot travels in clockwise direction around $O$ starting at $y$. Similarly, let $y_r$ be the nearest explored node on the boundary of $O$ if the robot travels in counter-clockwise direction. Node $y$ might be equal to either $y_l$ or $y_r$.

**Proposition 1** *Suppose that the robot hits a partially mapped obstacle $O$ at a node $y$ while executing Map-Obstacles. The following statements hold while $dist \leq R$.*

*a) 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.*

*b) 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 breakpoint.*

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

**Proof**: First assume that $y$ is equal to either $y_l$ or $y_r$. Say $y$ is equal to $y_l$. In line 24, $y_l$ is deleted from $B$. In the following execution of line 4, the robot moves along the unexplored boundary of the obstacle. If it reaches $y_r$, the robot moves back to $s$ (line 7) and deletes $y_r$ from $B$ (lines 10) provided that no unexplored boundary starts at $y_r$. Note that $new =$ false because if $new =$ true in the previous execution of line 17, then it was set to false in line 19.

Now assume that $y \neq y_l$ and $y \neq y_r$. Node $y$ is inserted into $B$ in line 26 and $new$ is set to true. In the following execution of line 4 the robot moves in one direction along the unexplored boundary. If it reaches, say, $y_l$, then in line 19 $P$ is set to be path back to $y$ because $new =$ true and $y$ is the most recently inserted breakpoint in $B$. In the next execution of line 22, the robot will try to reach $y$ (if the fuel suffices). Finally the robot tries to reach $y_r$ in the following execution of line 4. If it does reach $y_r$, then it moves back to $s$ in line 7. $\square$

**Lemma 3** *Suppose that the robot is located at $s$ and then executes code of Map-Obstacles. Whenever the robot has traversed $r + l$, $0 < l$, edges since the last visit to $s$ and has to yet moved back to $s$, it has traversed at least $l/2$ fresh edges.*

**Proof**: Initially, when the robot is located at $s$, let $b \in B$ be the most recently inserted breakpoint. The robot computes a shortest path from $s$ to $b$ assuming all obstacles are known. Whenever the robot hits a new obstacle at a node $y$ and can move around the boundary while $dist \leq R$, $y$ is first inserted into $B$ in line 26 and then deleted again in the following execution of line 10. In line 21, the robot computes a path from $y$ to $b$, which is still the most recently inserted breakpoint in $B$. Thus, if the path traveled so far has not reached a partially mapped obstacle, the statement of the lemma can be shown in the same way as Lemma 2.

If the path has reached a partially mapped obstacle $O'$ at a node $y$, then the robot is currently located between $y_l$ and $y_r$. This follows from Propositition 1 because after having explored edges of a partially mapped obstacle, the robot moves back to $s$. If $y = y_l$ or $y = y_r$, then all edges traversed by the robot on the boundary of $O'$ are fresh. If $y \neq y_l$ and $y \neq y_r$, then at least half of the edges are fresh. Let $\mathcal{O}$ be the set of obstacles hit by the robot before reaching $O'$. As in the proof of Lemma 3 we can show that when reaching $O'$, the robot has traversed at most $|P| + 2\sum_{O \in \mathcal{O}} |O| \leq r + 2\sum_{O \in \mathcal{O}} |O|$ edges. At least $\sum_{O \in \mathcal{O}} |O|$ of these edges are fresh. Here $P$ is the initial path from $s$ to $b$. Thus the total number of fresh edges is at least $\frac{1}{2}(r + l - (r + 2\sum_{O \in \mathcal{O}} |O|) + \sum_{O \in \mathcal{O}} |O| = l/2$. $\square$

**Lemma 4** *If a path $Q_i$ traverses $R$ edges, then it traverses at least $\alpha r/2$ fresh edges.*

**Proof**: Let $Q_i$ be an arbitrary path that traverses $R$ edges. We distinguish cases depending on the procedure in which the visit to $s$ at the beginning of $Q_i$ occurs.

1. The visit to $s$ occurs during a call of *Refuel*.

2. The visit to $s$ occurs during a call of *Map-Obstacles*.

**Case 1:** If the path $Q_i$ ends during the call of *Refuel*, then the lemma follows from Lemma 2. If the path ends after termination of *Refuel*, then the robot continues to explore edges on the exterior boundary or in calls of *Shoot-Rays* including executions of *Map-Obstacles*. All edges traversed when moving along the exterior boundary are fresh. Also, all edges traversed in *Shoot-Rays* are fresh by definition. If the robot executes calls of *Map-Obstacles*, then it traverses only fresh edges along the boundary of new obstacles. If at most $2r$ edges were traversed when *Refuel* terminates, then there is nothing to show. If $2r + l$, $0 < l$, edges were traversed at termination, then by Lemma 2 the total number of fresh edges in $Q_i$ is $R - (2r + l) + l/2 = (1 + \alpha)r - l/2 > \alpha r/2$.

**Case 2:** If the path ends during the call of *Map-Obstacles*, then the desired bound follows from Lemma 3. If $Q_i$ ends after the call and the robot continues with executions of *Shoot-Rays*, then the lemma can be shown using arguments of Case 1. It remains to study the case that after termination of *Map-Obstacles* the robot continues executing *Refuel*. Lemmas 2 and 3 imply that whenever the robot has traversed $3r + l$ edges, at least $l/2$ edges were fresh. If $Q_i$ ends in *Refuel*, then the desired bound follows. If the path ends after termination of *Refuel*, then the robot explores only fresh edges on the exterior boundary or in calls of *Shoot-Rays* including executions of *Map-Obstacles*. □

Consider an explored segment of a partially or fully mapped obstacle. The explored segment of a fully mapped obstacle is the entire boundary. Any such sequence can be decomposed into *fragments.* A fragment is a maximal sequence of consecutive edges that were explored successively on one of the paths $Q_i$, $1 \leq i \leq N$. Thus, with every fragment we can associate a unique $Q_i$.

**Lemma 5** *Let $O$ be a fully mapped obstacle whose boundary consists of of $k \geq 2$ fragments. Then there exist at least $\lceil \frac{k}{2} \rceil$ fragments for which the associated paths $Q_i$ traverse $R = (3 + \alpha)r$ edges.*

**Proof**: Consider the call of *Map-Obstacles* in which $O$'s boundary is explored. We examine the steps in which the fragments are built up.

When the first fragment is explored, the associated path $Q_i$ traverses $R$ edges: $O$ is hit at a node $y$ and the robot moves along $O$'s unexplored boundary. Since $O$'s boundary consists of at least two fragments, the robot does not reach $y$ again while $dist \leq R$. Thus $Q_i$ traverses $R$ edges. The lemma now follows for all obstacles consisting of $k = 2$ fragments.

Let $O$ be an obstacle whose boundary consists of $k > 2$ fragments. We show that, during the execution of *Map-Obstacles*, if an explored segment consists of $l$ fragments, then for at least $\lfloor \frac{l}{2} \rfloor + 1$ fragments the associated paths traverse $R$ edges. For a segment $\sigma$ consisting of one fragment, the statement clearly holds: The robot has hit $O$'s boundary at one of $\sigma$'s endpoints and then traversed boundary edges until $dist = R$.

There are two possibilities how an explored segment can grow. (1) An explored segment $\sigma$ is extended at one of the endpoints. (2) Two explored segments $\sigma_1$ and $\sigma_2$ are merged into a larger segment when the robot explores boundary edges between $\sigma_2$ and $\sigma_2$.

In case (1) there are again two possibilities how $\sigma$ can be extended. First, the robot can reach an endpoint of $\sigma$ and traverse unexplored edges starting from that node until $dist = R$. Secondly, the robot can hit $O$'s boundary at an unexplored node $y$ and then traverse unexplored boundary edges until an endpoint of $\sigma$ is reached. In this case, the robot will return to $y$ and explore boundary edges starting from $y$ until $dist = R$. In both cases the path traverses $R$ edges and in the extended segment $\sigma'$, for at least $\lfloor \frac{l}{2} \rfloor + 1 + 1 \geq \lfloor \frac{l+1}{2} \rfloor + 1$ fragments, the associated paths traverse $R$ edges.

For the analysis of case (2), let $\sigma_1$ and $\sigma_2$ be two segments consisting of $l_1$ and $l_2$ fragments. The path $Q_i$ joining $\sigma_1$ and $\sigma_2$ does not necessarily traverse $R$ edges. However, $\lfloor \frac{l_1}{2} \rfloor + 1 + \lfloor \frac{l_2}{2} \rfloor + 1 \geq \lfloor \frac{l_1+l_2+1}{2} \rfloor + 1$ for all combinations of odd end even $l_1, l_2$. Thus the above statement holds. □

**Lemma 6** *The number of paths $Q_i$ that traverse $R = (3 + \alpha)r$ edges is at least as large at the number of paths that traverse less than $R$ edges.*

**Proof**: A path only traverses less than $R$ edges if it hits a partially mapped obstacle during an execution of *Map-Obstacles* and explores edges between two explored segments. In this case the robot might return to $s$ before $R$ edges are traversed (see Proposition 1). Thus every path that traverses less than $R$ edges ends on the boundary of an obstacle and the associated fragment belongs to an explored segment with at least two fragments. Lemma 5 states that for any obstacle with at least two fragments, the number of fragments that represent paths of length $R$ is as least as large as the number of fragments that represent paths of length $< R$. The lemma follows because every path $Q_i$ explores at most one obstacle whose boundary consists of at least two fragments. □

**Theorem 4** *The Ray algorithm explores a grid with arbitrary obstacles using $O(|E|)$ edge traversals.*

**Proof**: During the execution of the *Ray* algorithm at most $4|E|$ fresh edges are traversed. There are $|E|$ edge traversals across new edges that are explored after the traversal. In executions of *Shoot-Rays*, every edge is traversed at most three times. An edge is traversed twice when the robot moves along vertical rays in line 4 or backtracks in line 13. An edge on the boundary of an obstacle can be traversed once more in lines 10 or 12. Lemma 4 implies that that there can be at most $\frac{2 \cdot 4|E|}{\alpha r}$ paths $Q_i$ that traverse $R$ edges. By Lemma 6 the number of paths that travese less than $R$ edges is bounded by the same number. Thus $\sum_{i=1}^{N} |Q_i| \leq \frac{16|E|}{\alpha r}(3 + \alpha)r \in O(|E|)$. □

## 4 Conclusions

In this paper we presented a lower bound of $\Omega(\sqrt{n})$ on the competitive ratio achieved by any online algorithm to explore a scene with $n$ axis-parallel rectangles, thus disproving a conjecture by Deng, Kameda, and Papadimitriou [5]. Our construction consists of $\Theta(\sqrt{n})$ stages each of which contains $\Theta(\sqrt{n})$ rectangles. The aspect ratio of the rectangles used in our construction is exponential in $\sqrt{n}$. This raises the interesting open question whether it is possible to explore scenes with a constant competitive ratio if the aspect ratio of the obstacles is bounded by a constant as, for instance, in scenes consisting of squares.

Secondly, we presented an algorithm for piecemeal exploration of grids with arbitrary (rectilinear) obstacles. Our algorithm uses $O(|E|)$ edge traversals if the robot is able to travel a distance of $2(3+\alpha)r$, for some constant $\alpha > 0$, between two consecutive visits to a start node $s$. Here $r$ is the maximum of all shortest path distances between $s$ and any other node in the graph. It is an open question whether a grid with arbitrary obstacles can be explored with $O(|E|)$ edge traversals if the robot can only travel a distance of $6r$ or less.

## References


[1] S. Albers and M. Henzinger. Exploring unknown environments. *Proc. 29th Annual ACM Symposium on Theory of Computing*, pages 416–425, 1997.

[2] B. Awerbuch, M. Betke, R. Rivest and M. Singh. Piecemeal graph learning by a mobile robot. *Proc. 8th Conference on Computational Learning Theory*, pages 321–328, 1995.

[3] M. Betke, R. Rivest and M. Singh. Piecemeal learning of an unknown environment. *Proc. 5th Conference on Computational Learning Theory*, pages 277–286, 1993.

[4] M. Bender and D. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. *Proc. 35th Symposium on Foundations of Computer Science*, pages 75–85, 1994.

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

[6] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Proc. 31st Symposium on Foundations of Computer Science*, pages 356–361, 1990.

[7] F. Hoffmann. Private communication, 1997.

[8] F. Hoffmann, C. Icking, R. Klein and K. Kriegel. A competitive strategy for learning a polygon. *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 166–174, 1997.

[9] F. Hoffmann, C. Icking, R. Klein and K. Kriegel. The polygon exploration problem: A new strategy and a new analysis technique. *Proc. 3rd International Workshop on Algorithmic Foundations of Robotics*, 1998.

[10] B. Kalyanasundaram and K. Pruhs. A competitive analysis of algorithms for searching unknown scenes. *Computational Geometry and Applications*, 3:139–155, 1993.

[11] J. Kleinberg. On-line search in a simple polygon. *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 8–15, 1994.

[12] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[13] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communication of the ACM*, 28:202–208, 1985.

[14] A.C.-C. Yao. Probabilistic computations: Towards a unified measure of complexity. *Proc. 17th Annual IEEE Symposium on Foundations of Computer Science*, pages 222-227, 1977.

## Appendix

We show how to handle some generalizations. If an edge in the graph belongs to the boundary of two obstacles, we simply keep a copy of the edge for each obstacle. When exploring boundary edges in *Map-Obstacles* the robot has to keep track, which copy of the edge is explored. The number of edge traversals increases by a factor of at most 2.

Suppose that start node $s$ is an arbitrary node in the grid. Let $L$ be the horizontal line through $s$ and let $L_1, \ldots, L_n$ be the segments of $L$ that lie entirely in the grid. First the robot explores the segments $L_1, \ldots, L_n$ and the exterior boundary of the scene. While doing so, it also explores obstacles $O_1, \ldots, O_k$ that intersect $L$.

The robot then explores the upper part of the scene by traversing rays in northern direction started from $L_1, \ldots, L_n$ and from the open segments in $O_1, \ldots, O_k$ that are to the north of $L$. Symmetrically, the robot explores the lower part of the scene by exploring rays in the southern direction. The total number of edge traversal is still $O(|E|)$.

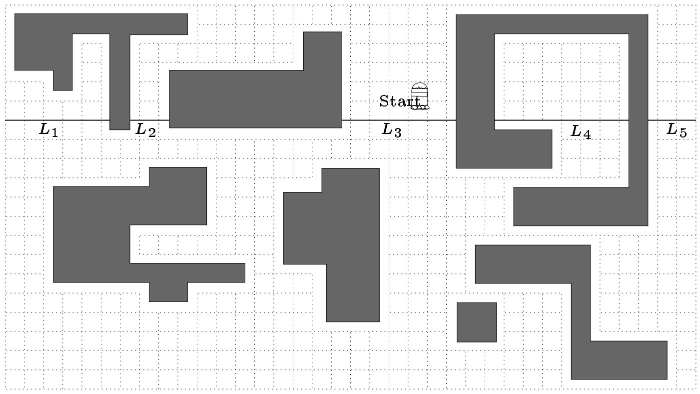


Figure 12: Start node $s$ is an arbitrary node.

In a similar way we can handle the case that the exterior boundary is not a rectangle or that obstacles touch the boundary. After the robot has explored the exterior boundary of the scene, it explores rays from all horizontal segments of the exterior boundary and from the open segments of the obstacles that touch the boundary.