NAVIGATING IN UNFAMILIAR GEOMETRIC TERRAIN

AVRIM BLUM *, PRABHAKAR RAGHAVAN †, AND BARUCH SCHIEBER ‡

Abstract. Consider a robot that has to travel from a start location s to a target t in an environment with opaque obstacles that lie in its way. The robot always knows its current absolute position and that of the target. It does not, however, know the positions and extents of the obstacles in advance; rather, it finds out about obstacles as it encounters them. We compare the distance walked by the robot in going from s to t to the length of the shortest (obstacle-free) path between s and t in the scene. We describe and analyze robot strategies that minimize this ratio for different kinds of scenes. In particular, we consider the cases of rectangular obstacles aligned with the axes, rectangular obstacles in more general orientations, and wider classes of convex bodies both in two and three dimensions. For many of these situations, our algorithms are optimal up to constant factors. We study scenes with non-convex obstacles, which are related to the study of maze-traversal. We also show scenes where randomized algorithms are provably better than deterministic algorithms.

Key words. Robot navigation, computational geometry, on-line algorithms

AMS subject classifications. 68Q25, 68T05, 52C05

1. Motivation and Results.

Practical work on robot motion planning falls into two categories: motion planning through a *known scene*, in which the robot has a complete map of the environment, and motion planning through an *unknown scene* in which an autonomous robot must find its way through a new environment (see, for example, [9, 13, 15, 21, 24] and references therein). Virtually all previous *theoretical* work ([32] and references therein) has focused on the former problem. Papadimitriou and Yannakakis [26] studied the latter problem, which is also the subject of this paper: the design and evaluation of strategies for navigation in an unknown environment. The unfamiliar environment may be either a warehouse or a factory floor whose contents are frequently moved, or a remote terrain such as Mars [30]. The design and evaluation of algorithms for such navigation is a natural algorithmic problem that deserves more theoretical study.

1.1. Model.

A *scene* $\mathcal{S}$ is a region (of $\mathbf{R}^2$ or $\mathbf{R}^3$) containing a start point s and a target t, together with a set of opaque, impenetrable, non-overlapping obstacles none of which contains s or t. Most of this paper will consider 2-dimensional scenes. The target t may be a point, or a polygon/polyhedron, or an infinite wall. To avoid certain degeneracies, we assume that a unit diameter circle (unit cube in three dimensions) can be inscribed in each obstacle; this guarantees that the obstacles have a certain minimum "thickness".

A point robot has to travel from s to t, and it knows both its current absolute position and the position of t. In walking towards t it must circumvent the obstacles in $\mathcal{S}$. The robot does not know the positions and extents of these obstacles in advance; rather, it finds out about obstacles as it encounters them. Where two obstacles touch, we assume that the robot can "squeeze" between them. Thus a scene that consists

* School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213. Email: avrimtheory.cs.cmu.edu. Part of this work was done while the author was visiting IBM T.J. Watson Research Center, and part while at MIT and supported by an NSF graduate fellowship.

† IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598. Email: praghwatson.ibm.com

‡ IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598. Email: sbarwatson.ibm.com

only of convex obstacles cannot have a non-convex obstacle composed of abutting convex obstacles.

The most natural mechanism for the robot to learn about a scene is vision: the robot discovers obstacles as they come into its view, and uses this information to decide how to proceed towards t. For simplicity of exposition we describe our algorithms assuming that when the robot first sees an obstacle it is given the shape, size and position of the obstacle (even though much of that obstacle may be invisible from where it stands). However, we show how many of our algorithms can be made to work with essentially the same upper bounds (up to a constant factor) under a considerably weaker assumption — a *tactile robot* that learns about obstacles only by bumping into them and moving along them. For this we use variants on the "doubling" strategies of Baeza-Yates *et al.* [1].

Let $R(\mathcal{S})$ be the total distance walked by a robot R in going from s to t in scene $\mathcal{S}$, and let $d(\mathcal{S})$ denote the length of the shortest (obstacle-free) path in the scene between s and t (because of the obstacles, this may be substantially larger than the Euclidean distance between s and t). Let $\mathcal{S}(n)$ denote the set of scenes in which the Euclidean distance between s and t is n. Following the lead of [26], we use as the figure of merit for the robot the *ratio*

$$\rho(R, n) = \sup_{\mathcal{S} \in \mathcal{S}(n)} \frac{R(\mathcal{S})}{d(\mathcal{S})},$$

and study its growth as a function of n.

For convenience, we put the scene in Cartesian coordinates, using "north"/"south" to denote the direction of increasing/decreasing y value, "east"/"west" for the direction of increasing/decreasing x value, and "up"/"down" for the direction of increasing/decreasing z value, respectively. In two dimensions, we also use "vertical" to mean parallel to the y axis, and "horizontal" to mean parallel to the x axis. The start point s is always assumed to be at the origin, and unless specified otherwise, we will assume the current scene belongs to $\mathcal{S}(n)$. Finally, we use $\log n$ to denote $\log_2 n$.

1.2. Summary of Results. For most of this paper we consider 2-dimensional scenes where t is a point and the obstacles are rectangles with sides parallel to the axes (rather than squares as in [26]). Surprisingly, even this problem turns out to be quite complicated. We solve this problem by breaking it into the following two subproblems.

The Wall Problem

Scenes in which t is an *infinite vertical line* and the obstacles are oriented rectangles. The goal is to reach a point on t of the robot's choosing.

The Room Problem

Scenes in which the obstacles are oriented rectangles that are confined to lie within a square "room". Here, s is a point on a wall of the room and t is the point at the center of the room. The robot can "squeeze" between any two obstacles or between the walls and any obstacle. This intriguing special case is of interest in its own right as a model for navigation in a bounded region such as a warehouse.

Section 2 describes an optimal algorithm for the wall problem. The algorithm achieves an upper bound of $O(\sqrt{n})$ on the ratio $\rho(R, n)$, matching the lower bound of [26]. To devise this algorithm we develop a general "sweep" paradigm that is fairly natural: a human lost in a strange city would probably do a similar search.

Section 3 considers the room problem. The algorithm for this problem achieves a ratio $\rho(R, n)$ that is $O(2\sqrt{3 \log n})$. Following and building upon our result, Bar-Eli *et al.* [2] have established a tight bound of $\Theta(\log n)$ on the ratio of deterministic algorithms for the room problem. The approach taken by the room problem algorithm is different from the one taken for the wall problem. Here, we develop a "caliper" method that pins the target down to lie within a sequence of advancing paths. Intriguingly, in the room problem the shortest path from s to t has length $O(n)$. To see this, suppose that s is the south-west corner of the room. So, the *greedy* path from the target t that travels due south if possible and otherwise due west will reach s and have length the L_1 distance between s and t. (If s is not in the corner, then by traveling along the room boundaries one can reach s at an additional constant factor cost.) In contrast, greedy paths from s are not guaranteed to go anywhere near t. Thus getting out of a room is easy, but getting in towards a small target seems to be hard.

Section 4 shows how to combine our solutions for the wall and room problems to obtain a tight bound of $\Theta(\sqrt{n})$ for point-to-point navigation in scenes consisting of oriented rectangular obstacles.

Section 5 describes how our algorithms work (with at worst a constant factor degradation in ratio) when the robot is tactile: it learns about obstacles by "feeling" them. In this case, our algorithms bump into obstacles and slide along their edges, in a manner reminiscent of *compliant* motion planning [7] in the context of navigation with a map.

Section 6.1 considers the room problem with arbitrary rectangular obstacles. We show that $d(\mathcal{S})$ can now be $\Omega(n^{3/2})$. Unlike the case of oriented rectangles the greedy path is no longer guaranteed to find an inexpensive way out of the room. For these scenes we give lower and upper bounds on $\rho(R, n)$.

Section 6.2 extends our algorithms for the room problem to the case of more general convex polygonal obstacles.

Section 7 gives extensions of our algorithm for the wall problem to three dimensions, and also for point-to-point navigation in three dimensions. Both of these algorithms work provided the obstacles are oriented rectangular cuboids, achieving optimal ratios. (A cuboid is a rectangular parallelepiped.)

In Section 8 we give a randomized algorithm for certain cases of the wall problem. We show that the (expected) ratio of our algorithm is $2^{O(\sqrt{\log n \log\log n})}$, which is much smaller than the corresponding deterministic lower bound. This demonstrates the power of randomization in navigation.

Section 9 deals with non-convex obstacles (and therefore mazes). We give a lower bound for randomized algorithms, and show that a deterministic algorithm of Rao *et al.* [28] meets this bound. The algorithm is memory-intensive, and so we offer an alternative algorithm that is very simple, memoryless, randomized and achieves the same upper bound in the plane.

We conclude with a list of some open problems in Section 10.

1.3. Related Theoretical Work. The ratio $\rho(R, n)$ is studied in Papadimitriou and Yannakakis [26], and independently by Eades, Lin and Wormald [14]. Papadimitriou and Yannakakis proved that when s and t are points in the plane, and all obstacles are squares, $\rho(R, n)$ is at least 1.5, and complement this with an algorithm attaining $\rho(R, n) \leq 1.5 + o(1)$ for all n. It is also shown in [14, 26] that when t is an infinite wall at distance n from s and the obstacles are oriented rectangles, then $\rho(R, n)$ is $\Omega(\sqrt{n})$. Coffman and Gilbert [12] study the performance of simple heuris-

tics in the presence of randomly placed obstacles. Kalyanasundaram and Pruhs [16] and Mei and Igarashi [22] consider scenes in which all obstacles have bounded aspect ratios. Klein [18] has given a small constant upper bound on the ratio for scenes that are *streets*, a class of simple polygons. Lumelsky and Stepanov [21] earlier gave a simple navigation algorithm that guarantees $R(\mathcal{S})$ to be bounded by $d(\mathcal{S})$ plus the sum of the perimeters of all obstacles, with no restrictions on the aspect ratios or the convexity of the obstacles. Their algorithm does not minimize the ratio ρ. Several papers (see [25, 28, 29] and references therein) give algorithms for building up a map of a scene by exploring it entirely. Maze-traversal has received considerable attention in the past in various papers [5, 19, 27], none of which considers the ratio metric. The reader is referred to [20] for a comprehensive survey of the results in these papers.

The ratio measure $\rho(R, n)$ has close connections to the *competitiveness* measure used in the study of on-line algorithms [6, 23, 31]; indeed, our problem resembles an on-line setting in which the obstacles encountered by the robot form a sequence of "requests", and we compare its total cost $R(\mathcal{S})$ to the "off-line cost" $d(\mathcal{S})$. It is therefore worth pointing out some key differences between the models: (a) In the navigation problem the robot has a definite target towards which it moves, while there is no such notion in on-line paging [31], for example. (b) The robot can move back and forth through the scene, re-visiting previously seen obstacles, thus having some control on the requests it encounters in the future. (c) Competitive analysis deals with request sequences of arbitrary (possibly infinite) length, whereas here we have a fixed number of obstacles in the scene. Thus we cannot cast our navigation problem in a standard on-line framework such as the server problem [23] or metrical task systems [6]. Nevertheless, the analogy with on-line algorithms proves useful in the study of randomized navigation (Section 8).

2. The Wall Problem. In this section we consider scenes in which t is an infinite vertical wall at distance n to the east of s, and the obstacles are rectangles whose sides are parallel to the axes. At the end of the section we show how to modify our algorithm to work also when t is not vertical. We call the *width* of an obstacle its length in the x-direction, and the *height* of an obstacle its length in the y-direction. To make the presentation clearer, we assume below that $\sqrt{n}$ is an integer. However, our algorithm and analysis can trivially be adapted to the general case.

We present an algorithm that achieves ratio $\rho(R, n) = O(\sqrt{n})$. This matches the lower bound proven in [26], so our algorithm is optimal up to constant factors.

The algorithm maintains four variables: the *window size*, W, a *threshold*, τ, a *sweep direction*, and a *sweep counter*. Initially, W is set to n, the sweep direction is south, and the sweep counter is set to zero. The threshold τ is always set to $W/\sqrt{n}$.

We begin with a high-level view of the algorithm and its analysis. The algorithm maintains a window of varying size around the x-axis. The robot makes $\sqrt{n}$ sweeps in directions alternating between north and south for each window size. Upon completion of these $\sqrt{n}$ sweeps the window size is doubled. Given a window of size W (which ranges from $y = +W/2$ to $y = -W/2$), the distance walked by the robot in sweeping is $O(W\sqrt{n})$. We show that the shortest path that cuts through all the $\sqrt{n}$ sweeps has length $\Omega(\sqrt{n}\tau) = \Omega(W)$. Let W_f be the window size at the time the robot reaches t. We prove that the total distance walked by the robot is $O(W_f\sqrt{n})$, while $d(\mathcal{S}) = \Omega(W_f)$.

We now describe the algorithm. Starting from point s, the robot travels due east until it either reaches t or hits an obstacle, say at (x, y). Below, we assume that the current sweep direction is south, the other case is symmetric. The next steps are

determined by the following rules:

Rule 1: If the distance to the nearest corner is less than τ, then the robot just goes "around" the obstacle. Specifically, it travels either south or north to the nearest corner, then east along the width of the obstacle to the opposite corner, and finally back along the height of the obstacle to the point $(x+w, y)$, where w is the width of the obstacle. (See Fig. 1(a).) From this point it continues to travel due east until it hits the next obstacle (or reaches t and stops).

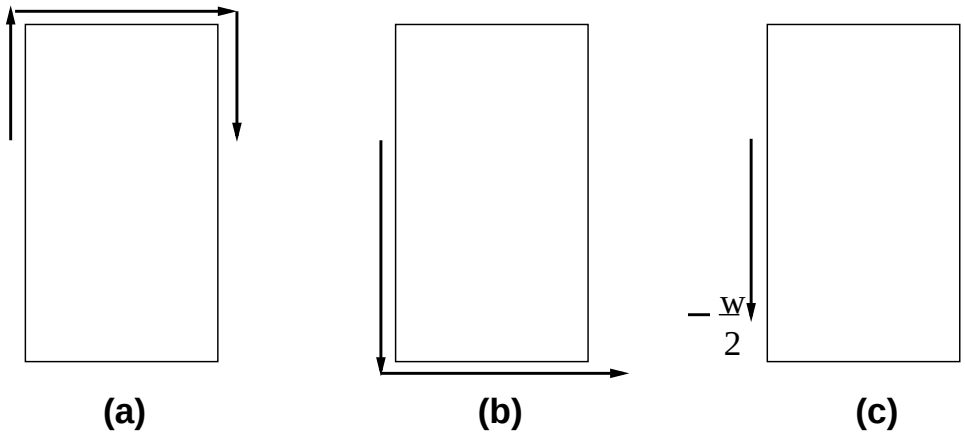

FIG. 1. *Going around an obstacle in the sweep algorithm*

Rule 2: If the obstacle extends past both sides of the window (i.e., its north edge has y-coordinate greater than $W/2$ and its south edge has y-coordinate less than $-W/2$), then the robot doubles the window width W and threshold τ, and resets the sweep counter to 0 and the sweep direction to south. Note that the ratio W/τ remains $\sqrt{n}$.

Rule 3: Otherwise (i.e., the distance to the nearest corner is more than τ, and the obstacle does not extend past both sides of the window), the robot travels south along the obstacle until either it hits the obstacle's south-west corner or it reaches the window boundary (y-coordinate $-W/2$). In the first case, the robot just continues due east to the next obstacle (or t). (See Fig. 1(b).) In the second case, the robot increments the sweep counter by 1 and flips the sweep direction. If the counter is greater than $\sqrt{n}$, the robot resets the counter to zero and doubles the window size and the threshold. (See Fig. 1(c).)

Let W_f be the window size at the time the robot arrives at t.

THEOREM 2.1. *The total distance walked by the robot is* $O(W_f\sqrt{n})$.

Proof. To prove that the distance is bounded by $O(W_f\sqrt{n})$, we divide the path taken by the robot into three components: (1) horizontal segments, (2) segments walked south and north "along" obstacles using Rule 1, and (3) segments walked south and north using Rule 3.

Notice that (i) the total distance walked east is $n \leq W_f\sqrt{n}$ since $W_f \geq n$, and (ii) since the width of each obstacle is at least one unit, the total distance walked south and north using Rule 1 is bounded by $2n\tau_f \leq 2W_f\sqrt{n}$, where τ_f is the final

threshold. It suffices to bound the third component as well. Fix a window size W. The distance walked by the robot using Rule 3 to complete one sweep is $O(W)$. Since the robot makes at most $\sqrt{n}+1$ sweeps for each window size, the total distance for a fixed window size is $O(W\sqrt{n})$. The window size is doubled each time it is changed, and thus the total distance traveled over all window sizes is also $O(W_f\sqrt{n})$. □

THEOREM 2.2. *The length of the shortest path from s to t, $d(\mathcal{S})$, is $\Omega(W_f)$.*

Proof. Since we are just interested in the length of the shortest path up to a constant factor, we may assume that the path consists only of horizontal and vertical segments. The length of the horizontal segments is clearly at least n. If $W_f = n$ then we are done. Also, if W_f is determined by rule 2 (there is an obstacle that extended past both sides of the previous window) we are done as well. So, assume neither of these is the case, which means the robot has completed at least $\sqrt{n}$ full sweeps for some window size $W \geq \frac{1}{2}W_f$. We now show that the vertical component of the shortest path has length $\Omega(W) = \Omega(W_f)$.

Consider a point on the shortest path with y-coordinate of maximum absolute value. If this absolute value is at least $W/2 - \tau$, then clearly the shortest path has length at least $W/2 - \tau = \Omega(W)$. Suppose that this is not the case. Given a shortest path, for each of the $\sqrt{n}$ sweeps, define its first entry point to be the first point on the shortest path whose x-coordinate is the same as the x-coordinate of the starting point of the sweep. Similarly, define its first exit point to be the first point on the shortest path whose x-coordinate is the same as that of the end point of the sweep. Note that since sweeps do not overlap in their x-coordinates, the exit point of sweep i appears before the entry point of sweep $i+1$. Thus, to lower bound the length of the shortest path we can add together for each sweep i the vertical components of the shortest path from the ith entry point to the ith exit point. Consider the obstacles touched by the robot during some sweep – let's say it is a "south" sweep – that require the use of Rule 3. Each such obstacle extends at least τ to the north of the southernmost point of the previous such obstacle. So, to travel from the entry point of the sweep (which is not between any two such obstacles) to the exit point of the sweep (also not between any two such obstacles) requires traveling a vertical distance at least τ. Since there are $\sqrt{n}$ sweeps, the total vertical component is at least $\tau\sqrt{n} = W$. □

COROLLARY 2.3. *Our sweep algorithm achieves a ratio of $O(\sqrt{n})$ for the wall problem, provided every obstacle is an oriented rectangle.*

A simple transformation of our algorithm allows it to achieve the same bounds even if the wall t is not vertical.

THEOREM 2.4. *The modified sweep algorithm below achieves a ratio of $O(\sqrt{n})$ for the wall problem with oriented rectangular obstacles, even if the wall is not vertical.*

Proof. The algorithm depends on the angle θ that t makes with the y-axis. Assume that $0 \leq \theta < \pi/4$ and that the wall runs from south-west to north-east. (The other three possibilities are analogous.) Here, n is the shortest Euclidean distance between s and t. We distinguish between two cases:

Case 1: $\sin\theta \geq 1/\sqrt{n}$. In this case the robot walks to t using the *greedy* east-south path from s (a path that travels due east if possible and otherwise due south). Observe that the length of this greedy path is the L_1 distance between s and the point of t that the path hits. The x-component of this L_1 distance is no more than $\sqrt{2}n$. Since $\sin\theta \geq 1/\sqrt{n}$ the y-component is bounded by $O(n^{1.5})$, implying $\rho(R,n) = O(\sqrt{n})$.

Case 2: $\sin\theta < 1/\sqrt{n}$. We run the sweep algorithm exactly as described above, until the first time the robot reaches a point (x_0, y_0) such that the point $(x_0, -W/2)$ is on or below t, where W is the width of the current window. Then, the robot walks to t

using the greedy east-south path. By Theorem 2.1 the distance walked by the robot until reaching (x_0, y_0) is $O(W\sqrt{n})$, and it is clear that the length of the greedy path from there to t is $O(n + W)$. By an argument identical to the one in the proof of Theorem 2.2, the shortest path from s to the vertical line $x = x_0$ has length $\Omega(W)$. By the requirement on θ the shortest path from s to t has length $\Omega(W)$ as well. □

3. The Room Problem. In this section we consider the *room problem*: scenes in which the obstacles are oriented rectangles confined to lie within a square room such that no obstacle touches the room walls; the point s is on the border of the room and t is in the center. (See Fig. 2.)

Later, we extend our results to rectangular rooms. Since travel along the room walls is "cheap", we may assume s is in the south-west corner of the room, and for convenience we let t have coordinates (n, n), so the distance from s to t is in fact $n\sqrt{2}$.

Define a *greedy* $\langle +x, +y \rangle$ path to be a path that travels due east if possible and otherwise due north. Similarly, define greedy $\langle +y, +x \rangle$ paths, $\langle +x, -y \rangle$ paths, and so forth, to be ones that travel in the first direction if possible and otherwise the second direction. A *brute-force* $\langle +x \rangle$ path is one that travels due east, going around obstacles in its way along the shorter direction, but otherwise maintaining a constant y coordinate. A *monotone* path from (x_1, y_1) to (x_2, y_2) is a path that does not both increase and decrease in any coordinate. For example, if $x_2 > x_1$ and $y_2 < y_1$, then the x coordinate will never decrease and the y coordinate will never increase. Notice that a greedy path is always monotone.

We now describe an algorithm achieving $R(\mathcal{S}) = O(n^{3/2})$, and thus $\rho(R, n) = O(\sqrt{n})$. An improvement that uses this algorithm recursively achieves $\rho(R, n) = O(2^{\sqrt{3 \log n}})$.

The algorithm maintains the following invariant at the start of each iteration: the robot knows of a monotone obstacle-free path from a point (x_0, n) to a point (n, y_0), where $0 \le x_0, y_0 \le n$. Furthermore, the robot is positioned on a point of this monotone path. We begin with $x_0 = y_0 = 0$, where the known path is just a path along the room borders. Each loop through the algorithm will increase either x_0 or y_0 by at least an amount $\sqrt{n}$, walking a distance of only $O(n)$. (If the value increased (x_0 or y_0) is within $\sqrt{n}$ of n, then it is increased only up to n.) Since each of x_0 and y_0 can be increased by this amount only $\lceil \sqrt{n} \rceil$ times, the total distance walked by the robot to reach t is $O(n^{3/2})$.

For this first version of the algorithm, let $m = \sqrt{n}$. We will describe the algorithm as if t were allowed to be inside an obstacle, in which case the goal is simply to reach the obstacle containing t; this will allow for easier recursive application.

Algorithm **Oriented-Room-Find**

Initialization: Set x_0 and y_0 to 0. Set the monotone path to be the path along the room boundary from (x_0, n) to (n, y_0).

Step 1: Define $\tilde{t}$ to be the point with x-coordinate $\min\{x_0 + m, n\}$ and y-coordinate $\min\{y_0 + m, n\}$. That is, unless we are close to t along some dimension, we have $\tilde{t} = (x_0 + m, y_0 + m)$. The goal of this step is to travel to some point t' not inside an obstacle that is to the northeast of $\tilde{t}$ and southwest of t inclusive. If no such point exists, we wish to travel to some point on the obstacle containing both $\tilde{t}$ and t.

For this (nonrecursive) version of the algorithm, we may reach t' as follows. First, traverse the monotone path to a point with y-coordinate equal to that

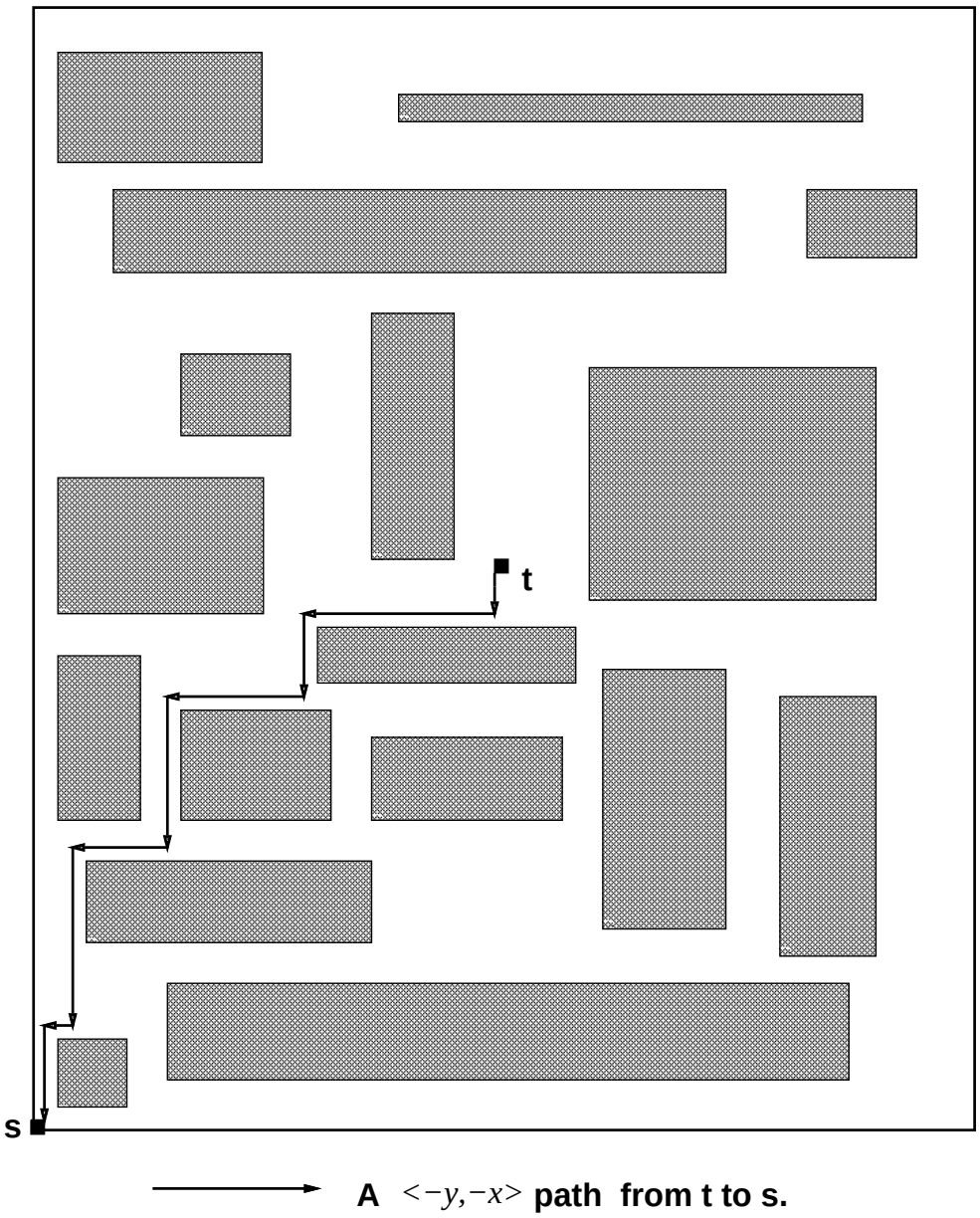

FIG. 2. *The Room Problem*

of $\tilde{t}$. Then, if this is to the west of $\tilde{t}$, travel in a brute-force $\langle +x \rangle$ path until $\tilde{t}$ is reached or an obstacle containing $\tilde{t}$ is first encountered. In the latter case, unless the obstacle contains both $\tilde{t}$ and t (and we are done), we can just follow the obstacle boundary to a point in the desired region.

Step 2: Make a greedy $\langle +x, +y \rangle$ path from t' until either the x or y coordinate equals n. If we are at t then halt. Otherwise, without loss of generality,

assume that the robot has traveled to the west of t, so the current coordinates are $(\hat{x}, n)$ for $\hat{x} < n$. Notice that $\hat{x} \geq x_0 + m$ since the path was greedy.

Step 3: Let $x_0 = \hat{x}$. Travel a greedy $\langle +x, -y \rangle$ path from $(\hat{x}, n)$ until either a point $(n, \hat{y})$ is reached (in which case let $y_0 = \hat{y}$), or the previous monotone path is hit. In either case we have a new monotone path from (x_0, n) to (n, y_0) with x_0 increased by at least m. If the greedy $\langle +x, -y \rangle$ path was such that it reached (n, y_0) and in doing so followed the border of a single obstacle, then we must be at an obstacle containing t and so are done. (Similarly, the robot would travel a greedy $\langle +y, -x \rangle$ path if it had hit to the south of target t in step 2.)
Now, go back to Step 1.

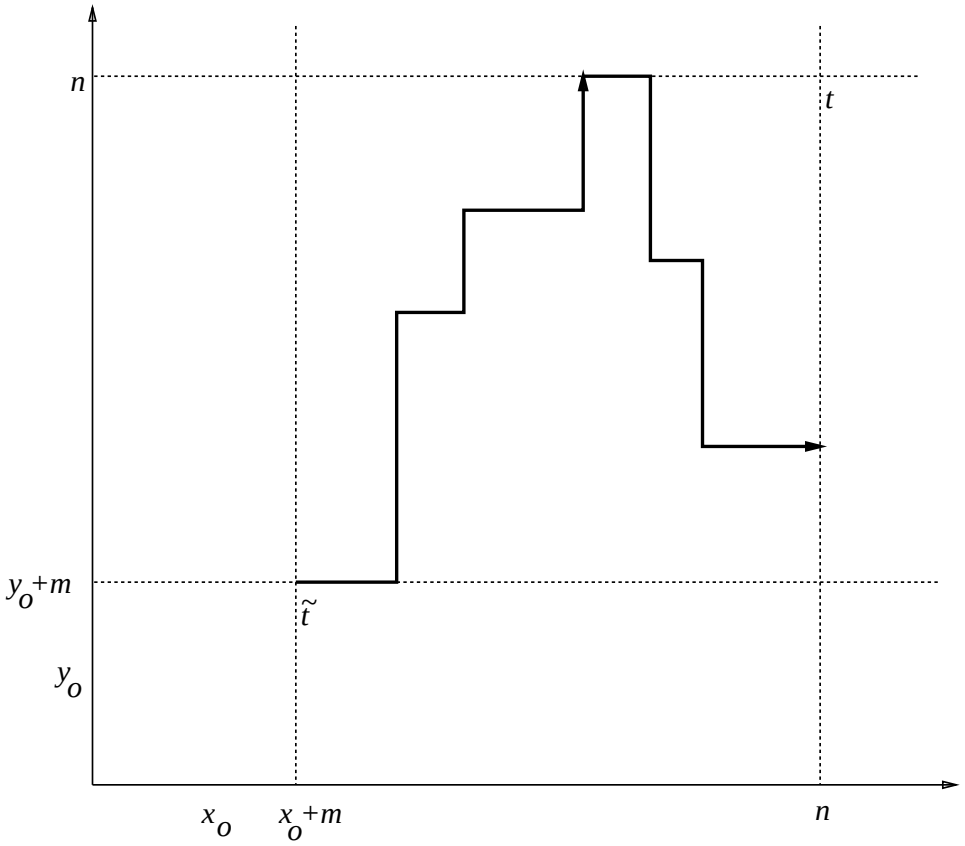

FIG. 3. *The algorithm for the room problem. Here* $\tilde{t} = t'$.

The distance walked performing Step 1 is $O(n)$ for traversing the monotone path and final obstacle, and $O(m^2) = O(n)$ for the brute-force path traveled, since the lower end of any obstacle encountered cannot extend below the monotone path (so there is at most $2m$ cost for every unit of progress made). Steps 2 and 3 together require traveling at most a distance $3n$ since each consists of a single greedy path, and in the two paths together at least one coordinate is non-decreasing. Since these

steps are iterated at most $2\lceil m \rceil$ times, the total distance traveled is at most $O(n^{3/2})$.

We can reduce the distance traveled to $O(n \cdot 2^{\sqrt{3 \log n}})$ by using algorithm Oriented-Room-Find recursively to reach $\tilde{t}$ (or an obstacle containing $\tilde{t}$) in Step 1, and optimizing the value of m. This works because to begin the algorithm needs only a monotone path of the form mentioned in the invariant, and not a true "room." In particular, there need not be any boundary "above" the obstacles at all. Define $T(n)$ to be the total distance traveled using this strategy to reach point $t = (n, n)$ (or a point on the boundary of the obstacle containing t if t is inside some obstacle), given that the robot is on a known monotone obstacle-free path from $(0, n)$ to $(n, 0)$. As a base case, if $n \leq 8$, say, we just use a brute-force path to reach t. So, the distance traveled at each iteration of Step 1 is at most $2n$ for traversing the monotone path, $T(m)$ for the recursive call, and $3n$ for following the boundary of the final obstacle encountered.[1] Since the number of iterations is at most $2\lceil n/m \rceil$, we can bound the total cost $T(n)$ by:

$$T(n) \;\leq\; 2\lceil n/m \rceil \left[T(m) + 8n\right], \quad \text{for } n > 8.$$

By substituting $m = n/(2^{\sqrt{3 \log n}})$, and using the inequality $\sqrt{x - k\sqrt{x}} \leq \sqrt{x} - \frac{k}{2}$ for $k > 0$, we get $T(n) \;\leq\; cn \cdot 2^{\sqrt{3 \log n}}$, for $c = 16(2 + \sqrt{2})$. We therefore have the following theorem.

THEOREM 3.1. *The algorithm for the room problem achieves* $\rho(R, n) = O(2^{\sqrt{3 \log n}})$.

If we consider a version of the room problem in which s and t are arbitrary points in the room, then the following strategy can be used to walk from s to t at a total cost that is $O(n2^{\sqrt{3 \log n}})$: simply walk from s out to a corner of the room, then use the above algorithm. Note, however, that in this case the length of the shortest path between s and t may be $o(n)$.

A generalization that will be used for the general point-to-point problem is when the room is rectangular with dimensions $2N \times 2n$, for $N \geq n$, and $t = (N, n)$. We use the same algorithm as for the square room, with one difference: we define point $\tilde{t} = (\min\{x_0 + mr, N\}, \min\{y_0 + m, n\})$ for $r = N/n$. (Again, if $n \leq 8$ we can just use a brute-force strategy to reach t traveling distance $O(N)$.) The value of m is optimized as follows. Define $T(n, r)$ to be the total distance traveled to reach point $t = (nr, n)$, given that the robot is on a known monotone obstacle-free path from $(0, n)$ to $(nr, 0)$. For a fixed value of m, the distance traveled at each iteration of Step 1 is at most $T(m, r) + 5nr$, while the distance traveled at each iteration of Steps 2 and 3 is at most $n + 2N \leq 3nr$. The number of iterations is at most $2\lceil n/m \rceil$, so we have:

$$T(n, r) \;\leq\; 2\lceil n/m \rceil \left[T(m, r) + 8nr\right] \quad \text{for } n \geq 8.$$

The substitution used above $[m = n/(2^{\sqrt{3 \log n}})]$ results in: $T(n, r) \;=\; O(rn \cdot 2^{\sqrt{3 \log n}})$.

Because we only needed a monotone path to start with, and not an entire room, we in fact have the following theorem.

THEOREM 3.2. *Given a monotone obstacle-free path between* $(0, n)$ *and* $(N, 0)$, *for* $N \geq n$, *the above algorithm will reach point* (N, n) *starting from that path with total cost* $O(N \cdot 2^{\sqrt{3 \log n}})$.

[1] The cost of traveling along the final obstacle can actually be amortized away at the expense of additional sentences of analysis.

4. Point-to-point Navigation. We combine the algorithms for the wall and room problems to obtain an algorithm for navigation in scenes where t is a point at (n, n) and the obstacles are oriented rectangles with no upper bounds on their extents.

The robot starts by taking a greedy $\langle +x, +y \rangle$ path from the start point s until it reaches a point s' with either x or y coordinate equal to that of t. Suppose that s' and t have the same y-coordinate. The robot now uses the sweep algorithm for the wall problem to travel to a point (n, y_0) with the same x coordinate as t. Without loss of generality assume $y_0 \geq n$. Notice that the path from s' to (n, y_0) taken by the robot in the sweep algorithm never decreases in the x direction, and that $y_0 - n$ is at most the final window width W_f. This path guarantees us that the *greedy* $\langle -y, -x \rangle$ path P from (n, y_0) will reach a point (x_0, n) with $x_0 \geq 0$ (in fact, x_0 is at least the x-coordinate of s'). Now we invoke the algorithm for the room problem (Theorem 3.2) to arrive at t using the monotone path P as the room walls.

We analyze the distance walked by the robot. The distance traveled using the algorithm for the wall problem is at most $O(W_f\sqrt{n})$, where W_f is the size of the last window considered. The size of the (rectangular) room then created is at most $n \times W_f$. So, using the algorithm of Theorem 3.2, the distance walked to reach t is $O(W_f\sqrt{n})$. By Theorem 2.2, the length of the shortest path from s' to t is at least cW_f for some constant $c > 0$. Now, if $W_f \leq 4n/c$ then we have an $O(\sqrt{n})$ ratio since $d(\mathcal{S}) \geq n$. If $W_f > 4n/c$, then since the length of the shortest path from s' to s is at most $2n$, $d(\mathcal{S}) \geq cW_f - 2n \geq \frac{c}{2}W_f$, so we also have an $O(\sqrt{n})$ ratio. We therefore have the following theorem.

THEOREM 4.1. *For two dimensional scenes $\mathcal{S}$ in which s and t are points and every obstacle is a rectangle whose sides are parallel to the axes our algorithm achieves a ratio of $\rho(R, n) = O(\sqrt{n})$.*

5. A tactile robot suffices. In this section we use a technique due to Baeza-Yates *et al.* [1] to demonstrate that all our algorithms given so far can be modified to work with essentially the same ratio bounds even if the robot is tactile: it learns about obstacles on bumping into them, and can infer the size of an obstacle only by moving along its boundary.

Suppose that the robot hits a side of a rectangular obstacle. Let d be the distance from its present position p to the nearest corner of the obstacle; it does not know d or the direction in which this nearest corner lies. The robot can reach this corner traveling a distance at most $9d + 2$ by applying the following "doubling" procedure suggested by Baeza-Yates *et al.* until a corner is reached. Walk along the side of the obstacle one unit in one direction, then turn back and walk two units past p in the other direction; turn back, and continue in this manner walking 2^{i-1} units past p on the ith iteration. If desired, in case a corner is reached which the robot is not certain is the nearest to p, the robot can simply walk an equal distance from p in the opposite direction to check. A simple analysis shows that the total length of the walk is at most $9d + 2$.

It remains to show that in each of the deterministic algorithms we have described, we can use this procedure to ensure that a tactile robot suffices (with a constant factor overhead in the ratio). In our sweep algorithm for the wall problem of Section 2, note that in negotiating an obstacle our decision is essentially based on the distance to the nearest corner of the obstacle. By using the above doubling procedure, we thus travel at most 9 times the distance that the visual robot does, plus a low-order term for the additive constant.

Next, consider the room problem. In our algorithm Oriented-Room-Find, the

only parts that required vision were the brute-force path, and finding a point between $\tilde{t}$ and t if $\tilde{t}$ was inside an obstacle in Step 1. Both can be handled with constant-factor overhead by the Baeza-Yates *et al.* procedure.

6. More general obstacle types.

6.1. Arbitrary Rectangular Obstacles. What if rectangular obstacles with sides not parallel to the axes are allowed in the room problem? We begin by proving two theorems that demonstrate the difference between scenes containing only oriented rectangular obstacles, and scenes containing arbitrary rectangular obstacles.

THEOREM 6.1. *For infinitely many n, there exist scenes $\mathcal{S}$ for the room problem containing rectangular obstacles whose sides are at arbitrary angles for which $d(\mathcal{S}) \geq \pi n^{3/2}/81$.*

Thus, the length of the shortest path between s and t is not always bounded above by the L_1 distance as in the oriented case.

THEOREM 6.2. *For any deterministic robot R, there exist scenes $\mathcal{S}$ for the room problem containing rectangular obstacles whose sides are at arbitrary angles for which $\rho(R, n) = \Omega(\sqrt{n})$.*

Thus, the upper bound for oriented rectangles cannot be achieved in this case.

Proof of Theorem 6.1. Consider $\lfloor n/27 \rfloor + 1$ circles centered at t, with radii $\lceil n/3 \rceil + 1 + 9i$, $i = 0, \ldots, \lfloor n/27 \rfloor$. Inscribe in each a regular $\lfloor \sqrt{n} \rfloor$-gon, aligning all these polygons. Rotate all the polygons inscribed in circles of radii $\lceil n/3 \rceil + 9i$ for even i by an angle $\pi/\sqrt{n}$. Each edge of each polygon can now be replaced by a rectangular obstacle of unit width (in the radial direction) and length very nearly the length of that edge. The length of each obstacle is at least $2\pi\sqrt{n}/3$. Now, any obstacle-avoiding path between s and t has to walk a distance of at least $2\pi\sqrt{n}/3$ going from a vertex of the polygon (i.e., gap between the obstacles) on the circle with radius $\lceil n/3 \rceil + 18i$ to a vertex on the circle with radius $\lceil n/3 \rceil + 18i + 18$, for $0 \leq i < n/54$. □

Proof of Theorem 6.2. Consider the scene described in the proof of Theorem 6.1. We allow a (deterministic) robot to walk from s to t. We now remove from the scene any obstacle not touched by the robot. Let T be the number of obstacles it touches. There is a constant c_1 such that the distance walked by the robot between touching a corner of every *fourth* new obstacle is at least $c_1\sqrt{n}$. This sums to a distance of at least $c_1 T\sqrt{n}/4$. The total area of the obstacles touched by the robot is bounded from above by $2c_1 T\sqrt{n}$. Thus there exists an angle $2i\pi/\sqrt{n}$, $1 \leq i \leq \sqrt{n}$, such that the path from t to s, given by staying on the radius at this angle and going "around" obstacles encountered is of distance at most $c_2 T$, for some constant c_2. This implies that $\rho(R, n) = \Omega(\sqrt{n})$. Notice that the lower bound holds only if the robot is tactile and cannot use any visual information. To make the lower bound work in the case of a robot that uses visual information we use a slightly different construction together with a technique given in [26].

Again consider $\lfloor n/27 \rfloor + 1$ circles centered at t, with radii $\lceil n/3 \rceil + 1 + 9i$, $i = 0, \ldots, \lfloor n/27 \rfloor$, and inscribe in each a regular $\lfloor \sqrt{n} \rfloor$-gon, aligning the polygons. This time, however, rotate all the polygons inscribed in circles of radii $\lceil n/3 \rceil + 18i$ and $\lceil n/3 \rceil + 18i + 9$ for even i by an angle $\pi/\sqrt{n}$. For each i call the polygons inscribed in circles of radii $\lceil n/3 \rceil + 18i$ and $\lceil n/3 \rceil + 18i + 9$ a *layer*. Each edge of the inner polygon of each layer is replaced by a rectangular obstacle as above, except that it has thin openings spaced at unit distances. The outer polygon has similar obstacles in all edges but one, which has a solid rectangular obstacle with no holes. The holes in each inner polygon are out of alignment with the corresponding holes in the outer polygon; thus the robot cannot see through any layer.

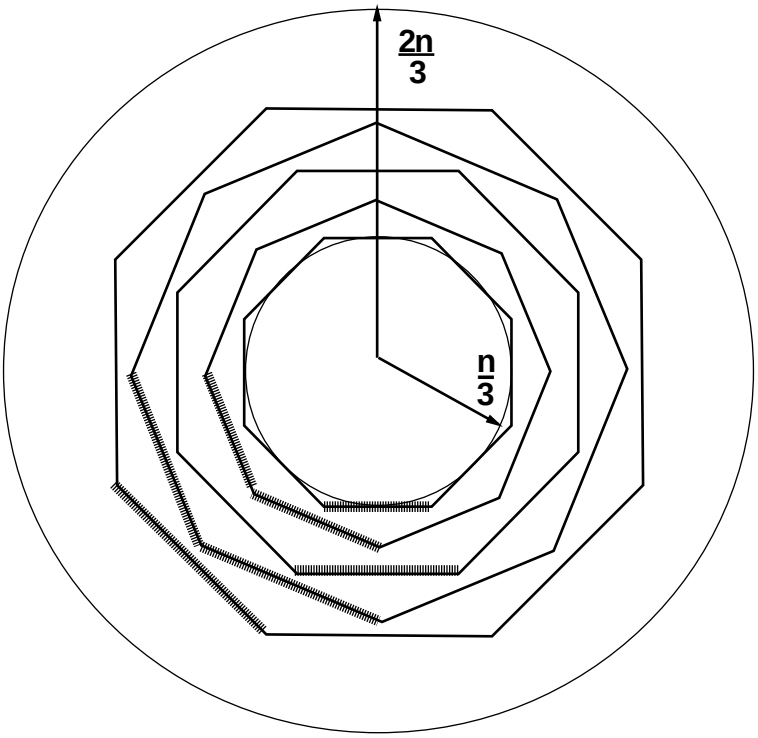

FIG. 4. *The lower bound in Theorem 6.2. The obstacles touched by the robot are shaded.*

We run the robot algorithm layer by layer, and make the first obstacle seen by the robot in each outer polygon a solid obstacle. By the same argument as above, it follows that the robot walks $\Omega(\sqrt{n})$ between every fourth layer. □

We now turn to upper bounds. Define the angle of a rectangle to be the angle of its longest edge with the x axis. We first describe a modification of algorithm Oriented-Room-Find to handle not just obstacles of angles of 0 and $\pi/2$, but obstacles angled in the range $[0, \pi/2]$ as well. Note that we do *not* allow obstacles angled in the remaining range of $(\pi/2, \pi)$. Then, we describe how this new algorithm can be modified for scenes $\mathcal{S}$ where there is a fixed known excluded range (d_1, d_2) of angles (for example, $d_1 = \pi/5$ and $d_2 = \pi/4$). Let $\tilde{n} = n/\alpha$, where $\alpha = d_2 - d_1$. Our algorithm achieves $R(\mathcal{S}) = (\tilde{n} \cdot 2^{3\sqrt{\log \tilde{n} \log\log \tilde{n}}})$. The length of the shortest path in such scenes is $O(n)$. We remark that for "practical" cases it may be enough to consider scenes where there is a known excluded range, especially when the number of different angles is small. Finally, we give a randomized algorithm that achieves $\rho(R, n) = \sqrt{n} \cdot 2^{O(\sqrt{\log n \log\log n})}$ regardless of the angles of the obstacles.

Suppose that the obstacles are angled in the range $[0, \pi/2]$. We describe our algorithm for this case in two steps. We first show that we need only consider the case where obstacles are zero-width line segments angled in the range $[0, \pi/2]$ such that at most some constant number of obstacles cross any line of length one. We then give an algorithm for that special case.

The idea for translating into the zero-width case is to view each obstacle as either

two or four line-segment obstacles consisting of its edges that are in the legal angle range. (All four edges are in the legal range only if the obstacle is oriented.) One can easily verify that this implies a constant upper bound on the number of obstacles that cross any given line segment of length one. Let us first assume that when the robot touches an obstacle, it is given its entire description (as has been our standard model). So, the robot can simulate an algorithm for the zero-width case, and if the simulated algorithm enters a real obstacle, the robot just waits until that algorithm exits and then meets it at the exit point and continues. By going around the obstacle in the shortest way, the (actual) robot travels at most twice as much as the simulated algorithm. To do the simulation with a tactile robot, notice that when an obstacle edge is touched, the robot can determine whether it should be treated as a zero-width obstacle or as empty space, based on its angle. If the obstacle is to be treated as empty space, the robot can use the Baeza-Yates *et al.* technique to find the point where the simulated algorithm would exit the (actual) obstacle with only a constant factor additional cost. The key point here, and in the previous case, is that the longer edges of an obstacle are never treated as empty space.

We now describe the algorithm for the zero-width case. The reason for reducing to this case is that since every obstacle edge now has angle in the range $[0, \pi/2]$, we can perform greedy $\langle +x, +y\rangle$ and $\langle +y, +x\rangle$ paths. The reason we cannot immediately use Algorithm Oriented-Room-Find, however, is that we can no longer make the greedy $\langle +x, -y\rangle$ and $\langle +y, -x\rangle$ paths required in Step 3. Instead, we will replace that portion of the algorithm with a less-efficient binary search strategy.

More precisely, let us say our start point is (x_0, y_0) and t is at (n, n). In contrast to Oriented-Room-Find, our invariant will be that we have *two* monotone paths: a $\langle +y, +x\rangle$ path from (x_0, y_0) to some point to the west of t (i.e., a point (x, n) where $x \leq n$), and a $\langle +x, +y\rangle$ path from (x_0, y_0) to some point to the south of t. As in Oriented-Room-Find, we begin by recursively (or using brute-force if the distance to t is sufficiently small) traveling to a temporary point t' defined as in that algorithm, at distance $O(m)$ from s. Now, in place of Steps 2 and 3 of that algorithm, we will instead use a binary search (described below) to find a point with either the same x coordinate or the same y coordinate as t' (and to the northeast of (x_0, y_0)) with the following property P: the greedy $\langle +x, +y\rangle$ and $\langle +y, +x\rangle$ paths from this point pass to the south and west of t respectively. Thus we will maintain our invariant, increasing either the x or y coordinate of the new "start point" as in Oriented-Room-Find.

We now show how to find the desired point. First, if t' is such a point we are done. Otherwise, suppose that both greedy $\langle +x, +y\rangle$ and greedy $\langle +y, +x\rangle$ paths starting at t' hit points to the south of t. (The other case is analogous.) Let s' be a point with the same y coordinate as t' on the $\langle +y, +x\rangle$ path from (x_0, y_0) given by our invariant. Since the $\langle +y, +x\rangle$ path from s' hits a point to the west of t, if the $\langle +x, +y\rangle$ path from s' hits a point to the south of t, then we are done as well. Otherwise, we travel to a point t'' halfway between s' and t'—using the same procedure as that used to reach t'—and examine the $\langle +x, +y\rangle$ and $\langle +y, +x\rangle$ paths from t''. (If t'' as defined is inside an obstacle, we examine the two points to the west and east of t'' on that obstacle boundary.) Depending on the outcomes of these greedy paths, we either halt with success or continue the binary search with a new t''' and so on. We stop the binary search when either success is discovered or the interval under consideration has length at most 1. So, at most $\lceil \log n \rceil$ iterations of the binary search will be made. If the binary search stops because the interval remaining is too short, a point with property P can be found easily by traveling from the west endpoint to the east endpoint of the

interval, and each time an obstacle is hit (this can happen at most a constant number of times), testing it for property P before going around the obstacle, which costs only $O(n)$. This strategy succeeds because if two points a and b have the same y-coordinate and there is either no obstacle between them or both are at the boundary of the same obstacle then the $\langle +x, +y\rangle$ path from the leftmost point intersects the $\langle +y, +x\rangle$ path from the rightmost point.

We get that the total distance is given by:

$$T(n) \;\leq\; \lceil 2n/m\rceil\,\lceil \log n\rceil\,[T(m) + cn]\,, \quad \text{for some constant } c.$$

Substituting $m = \left\lceil n/2\sqrt{\log n \log\log n}\right\rceil$ yields

$$T(n) \;=\; O(n \cdot 2^{3\sqrt{\log n \log\log n}}).$$

This strategy can be used for a smaller range (d_1, d_2) of excluded angles by just performing a rotation and a coordinate transformation on the space. Essentially, instead of writing t as $n\vec{x} + n\vec{y}$ for orthogonal unit vectors $\vec{x}$ and $\vec{y}$, we may write t as $(n'\vec{d_1} + n''\vec{d_2})$, where $\vec{d_1}$ and $\vec{d_2}$ are unit vectors in the d_1 and d_2 directions. It is not difficult to see that both n' and n'' are $O(n/\alpha)$, where $\alpha = d_2 - d_1$. Let $\tilde{n} = n/\alpha$. The performance of the previous algorithm after the transformation is $R(\mathcal{S}) = O(\tilde{n} \cdot 2^{3\sqrt{\log \tilde{n} \log\log \tilde{n}}})$, since the lengths are changed by at most a factor of $1/\alpha$.

THEOREM 6.3. *There is a deterministic algorithm for the room problem with an excluded angular range of size α that achieves $R(\mathcal{S}) = O(\tilde{n}2^{3\sqrt{\log \tilde{n} \log\log \tilde{n}}})$. Here $\tilde{n} = n/\alpha$.*

Consider now the general case where the angles of the obstacles may be in any range. A simple pigeonholing argument implies that a constant fraction of the ranges $[i\pi/\sqrt{n}, (i+1)\pi/\sqrt{n}]$, for $0 \leq i < \sqrt{n}$ have the property that the total perimeter of the obstacles angled in this range is no more than $2/\sqrt{n}$ of the total perimeter. To bound the total perimeter note that from our assumption that a unit circle can be inscribed in each obstacle it follows that the perimeter of an obstacle is always less than four times its area. Since the total area of all obstacles is at most n^2, the total perimeter of obstacles in such a range is $O(n^{3/2})$.

Consider a randomized algorithm that first guesses such a range. It then applies the above algorithm assuming that there are no obstacles with angles in this range. On actually encountering any obstacle in this range, it just goes around the obstacle at cost at most the perimeter of the obstacle. From the definition of the "forbidden angle range" it follows that on any given greedy path, the robot will go around any such obstacle at most once. Therefore, the expected total distance walked by this algorithm is given by the recursion given above where $\tilde{n} = n^{3/2}$, and a constant times $n^{3/2}$ is added to the $c\tilde{n}$ term (which remains $O(\tilde{n})$). Thus, we obtain the solution $T(n) = n^{3/2} \cdot 2^{O(\sqrt{\log n \log\log n})}$.

THEOREM 6.4. *There is a randomized algorithm achieving a ratio $\sqrt{n}\cdot 2^{O(\sqrt{\log n \log\log n})}$ for the room problem provided every obstacle is a rectangle within which a unit circle can be inscribed.*

6.2. Arbitrary convex polygons. We now describe how our randomized algorithm for the room problem can be extended to handle arbitrary convex polygons provided that a unit circle can be inscribed in each obstacle, and that the entire description of an obstacle is given to the robot when that obstacle is touched. (The

only part of the description of the obstacle that is required is the angle its longest diagonal makes with the x-axis.) We do not have a solution for the wall problem with arbitrary convex obstacles, and thus no solution for point-to-point navigation with convex obstacles.

Define the *angle* of a convex polygonal obstacle to be the angle its longest diagonal makes with the x-axis. The idea for the conversion is that each time the robot encounters an obstacle, it picks a longest diagonal D and treats that obstacle as a collection of line segments parallel to D. In particular, it imagines a line segment at D, and then additional segments (if any) parallel to and at distance 1,2,3, etc. from D, each as long as possible to be still contained within the obstacle. It then feeds this collection of line segments to the algorithm for rectangular obstacles. As in the case for unoriented rectangular obstacles, suppose the line-segment algorithm wishes to travel a path along line segments that leads through one of the convex obstacles: say the path is between points a and b on some obstacle's border. The robot then simply travels the shortest path from a to b along the obstacle boundary. Since the line segments are parallel to the longest diagonal of the obstacle and the obstacle is convex, we are guaranteed that the shortest path along the obstacle between a and b is at most a constant multiple of the straight-line path.

We note that in case there is a fixed known excluded range of angles, then the algorithm of Theorem 6.3 can be extended as well.

7. Extensions to three dimensions. This section summarizes extensions of our techniques to three dimensions. We begin by extending our study of the wall problem to three dimensions, and then extend our optimal algorithm for point-to-point navigation to three dimensions.

7.1. The wall problem in three dimensions. Suppose that t is an infinite plane perpendicular to the x-axis at distance n from the origin s. We begin by extending the lower bound of [26] to three dimensions, showing a lower bound of $\Omega(n^{2/3})$. We then give a generalization of the two-dimensional sweep algorithm that achieves a matching upper bound.

THEOREM 7.1. *For any deterministic robot, there are scenes $\mathcal{S}$ of the three-dimensional wall problem for which $\rho(R, n) = \Omega(n^{2/3})$.*

Proof. To prove the lower bound, it will be convenient to assume a tactile robot. Using the technique of [26] used in Section 6.1, this proof can be extended to robots with visual capabilities.

As the robot walks in the direction of t, the adversary places obstacles as follows. Each obstacle is a cuboid whose cross-section parallel to the yz plane is a square of side $n^{2/3}$, and whose width in the x direction is one. Whenever the robot first reaches x-coordinate i, for each $i \in \{0, 1, \ldots, n-1\}$, a cuboid is placed directly in front of it. Thus the robot must travel a distance at least $\frac{1}{2}n^{2/3}$ perpendicular to the x-axis in order to advance one unit parallel to the x-axis. Thus $R(\mathcal{S}) \geq \frac{1}{2}n^{5/3}$.

We now show that $d(\mathcal{S}) \leq 3n$. Since the cross-sectional area of each cuboid is $n^{4/3}$, by the pigeonhole principle there is a line ℓ parallel to the x-axis with the following properties: (a) its distance from the x-axis is at most n; (b) it cuts at most $n^{1/3}$ cuboids. Consider a path that starts from s and first goes to the leftmost point of ℓ. It then goes along ℓ parallel to the x-axis, traveling around each cuboid it encounters.

The distance from s to the leftmost point of ℓ is at most n. The distance traveled parallel to the x-axis is also n. The total perpendicular distance traveled in circumventing the cuboids cut by ℓ is at most $n^{1/3} \times n^{2/3} = n$. So, $d(\mathcal{S}) \leq 3n$ and the ratio

$\rho(R, n) = \Omega(n^{2/3})$. □

We now give an algorithm that matches this lower bound to within a constant. At a high level, the algorithm can be viewed as an extension of the two-dimensional sweep algorithm. The window, which in the plane was the region between two lines parallel to the x-axis, now becomes a cylinder whose axis is the x-axis. The radius of this cylinder is initially n, and is subsequently increased at certain points. The sweep used in the plane is now replaced by a spiral about the x-axis.

For simplicity, we first describe the algorithm as if every obstacle were a cylinder of circular cross-section, its axis parallel to the x-axis, and its center placed directly in front of the robot. The radii and lengths of these obstacles could vary. Following the analysis of this simple case, we outline the extension of the algorithm to more general obstacles.

Consider a point orbiting around a fixed point, with the radius of the orbit increasing linearly with angular position at a rate of D units for every 2π radians of angular position. We call the path of the moving point a *spiral*, and D the *spacing* of this spiral. (See Fig. 5.)

Our algorithm begins with $W = n$; at all times, $\tau = W/n^{1/3}$. Consider a spiral whose center is on the x-axis and whose orbits lie in a plane perpendicular to the x-axis. The spacing will be $\tau/3$. Thus there are at most $3n^{1/3}$ orbits in the spiral within the current window.

The y and z coordinates of the robot will always lie on such a spiral. Analogous to the sweep direction in the plane (north or south), the robot now maintains a spiral direction that is either "outwards" or "inwards" along the spiral. On encountering a (cylinder) obstacle directly in front of it, the robot first checks if the radius of the cylinder exceeds τ. If not, the robot "goes around" the cylinder, retaining its current yz coordinates. If on the other hand the radius does exceed τ, the robot proceeds to the nearest point p on the spiral along its current spiral direction (outwards or inwards) that is not covered by the cylinder, proceeding as far along the x direction as it can in the process. If this nearest point lies at a distance W' from the x-axis that exceeds W, the robot increases W to $2W$, resets the sweep counter to zero, and proceeds to begin a new spiral inwards from this point. Whenever the robot completes an inward spiral by reaching the x-axis or an outward spiral by reaching a point at distance W from the x-axis, it increments the sweep counter. Whenever the sweep counter reaches $n^{1/3}$, the robot doubles W (and τ), resets the sweep counter to zero and continues.

THEOREM 7.2. *The spiral algorithm achieves a ratio of $O(n^{2/3})$, provided every obstacle is a cylinder whose center is directly in front of the robot when it is first encountered.*

Proof. The analysis is essentially identical to that in Theorems 2.1 and 2.2. Let W_f be the final window radius. Since there are at most $3n^{1/3}$ orbits in each spiral, the distance walked by the robot in the last completed spiral is $O(n^{1/3}W_f)$. Since there are at most $\lceil n^{1/3} \rceil$ complete spirals in each window the total distance walked by the robot is $O(n^{2/3}W_f)$.

We show that the length of the shortest path is $\Omega(W_f)$. First, we may assume that the robot has completed $\lceil n^{1/3} \rceil$ spirals for some window size $W \geq \frac{1}{2}W_f$, or else we are immediately done (this is by the same reasoning used for the two-dimensional wall problem). Consider a point on the shortest path that is farthest from the x axis. If the distance of this point to the x axis is at least W, then clearly the shortest path has length at least W; so, we may assume that this is not the case.

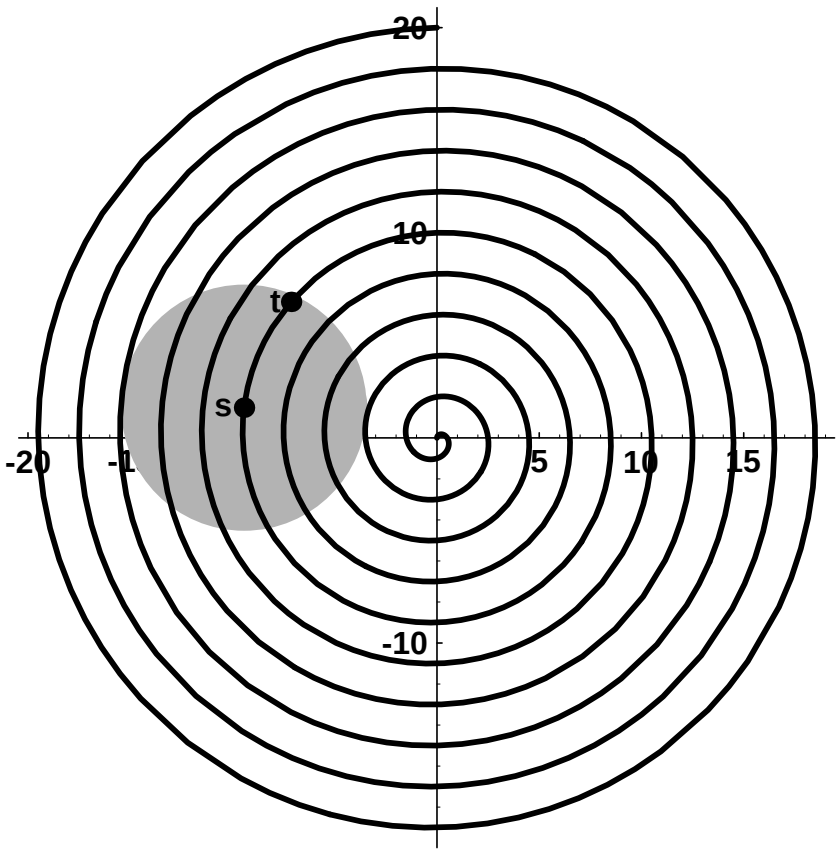

FIG. 5. *A projection of a spiral with radius 20 and spacing 2, overlaid with a projection of a cylinder of radius 6. The point s is the projection of the point where the robot encountered the obstacle, and point t is the projection of the nearest point on the outward spiral that is not covered.*

Given a shortest path, for each of the completed spirals, define its first entry point to be the first point on the shortest path whose x-coordinate is the same as the x-coordinate of the starting point of the spiral. Similarly, define its first exit point to be the first point on the shortest path whose x-coordinate is the same as that of the end point of the spiral. As in the two-dimensional problem, the exit point of spiral i appears before the entry point of spiral $i+1$. So, we need only show that for each spiral in some window size W, the yz plane component of the shortest path from the entry point to the exit point of the spiral is $\Omega(\tau)$. This will imply that the total yz

plane component of the shortest path is $\Omega(\tau n^{1/3}) = \Omega(W)$. So, imagine projecting the completed spiral onto the yz plane, projecting all the cylinders encountered in that spiral onto circles in the yz plane. Observe that every point on the spiral is at distance at most $\tau/2$ from the center of one such circle of radius at least τ. Since the orbits of the spiral are at distance $\tau/3$ from one another, for any point of distance at most W from the origin there exists a circle such that the distance of this point from the periphery of the circle is $\Omega(\tau)$. In particular, this also holds for the projection of the entry point of the sweep. Thus the yz component of any path from the entry point to the exit point must be $\Omega(\tau)$. □

The extension to the case of general cylindrical obstacles is similar. We define a general cylindrical obstacle to be one for which there is a simple closed curve C in the yz-plane such that the obstacle's intersection with any plane perpendicular to the x-axis, when translated to the yz-plane is either empty or it is C and its interior. As long as the robot moves in the positive x-direction, it will hit an obstacle only at some point of its unique "west face" in the yz-plane. On encountering such an obstacle, the robot measures the shortest distance from its present position to a point p on the spiral not touched by the obstacle. If this quantity is less than τ it uses this shortest path to circumvent the obstacle and retain its yz coordinates. Otherwise, it goes to p and proceeds as far along the x direction as it can. The analysis is very similar to the case of unit-height cylinders.

THEOREM 7.3. *For three dimensional scenes $\mathcal{S}$ with general cylindrical obstacles and in which s and t are points our spiral algorithm achieves a ratio of $\rho(R, n) = O(n^{2/3})$ for the wall problem.*

7.2. Point-to-point navigation in three dimensions. We now give an upper bound for point-to-point navigation in three dimensions that matches the lower bound to within a constant factor provided every obstacle is a cuboid whose sides are parallel to the axes. As in two dimensions, our upper bound for point-to-point navigation comes from combining an algorithm for point-to-plane navigation and another for the room problem. However, in the three dimensional case it suffices to combine the three dimensional wall algorithm with the two-dimensional room algorithm to obtain a three-dimensional point-to-point navigation algorithm. For simplicity of analysis, we assume all obstacles have vertices at integral coordinates. However, our algorithm would still work provided a unit cube can be inscribed within every cuboid in the scene.

Suppose without loss of generality that the x, y, and z coordinates of s are less than those of t. The algorithm consists of three stages. In the first stage the robot reaches a point s'', such that at least two of its coordinates are the same as t. This is done as follows. The robot starts by taking a greedy $\langle +x, +y, +z\rangle$ path until one of the three coordinates is the same as t. Call this point s' and w.l.o.g. say the y-coordinates of s' and t are the same. Next, fixing the y-coordinate (i.e., staying in the xz-plane of point s'), the robot takes a greedy $\langle +x, +z\rangle$ path from s' until one of the other two coordinates is the same as t. The endpoint of this path is the desired point s''. W.l.o.g. assume that the y and z coordinates of s'' are the same as t and let $n_x \leq n$ be the distance between s'' and t. The total distance walked in the first stage is $O(n)$.

In the second stage the robot uses the three dimensional wall algorithm from s'' to reach a point t' with the same x-coordinate as t. See Fig. 6. The total distance walked in this step is at most $O(n^{2/3})$ times $d(\mathcal{S})$. Let n_w be the distance between t' and t.

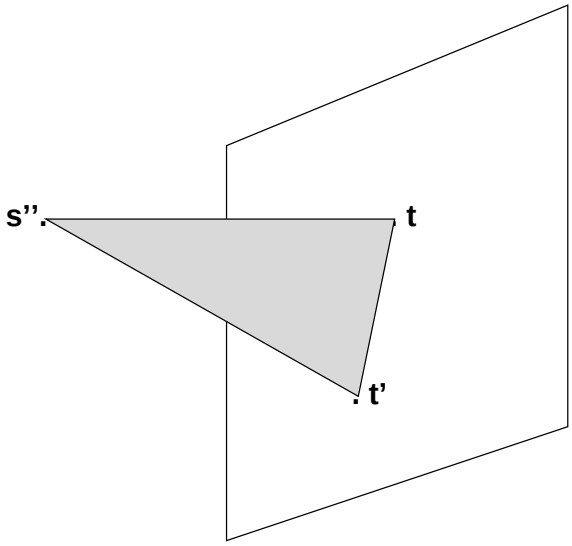

FIG. 6. *The plane that contains the points s'', t' and t.*

Assume that t' is not t (otherwise we are done). Consider the plane that contains the three points s'', t' and t. (See Fig. 6.) In the third stage the robot will stay in this plane. Notice that since all the obstacles are cuboids, the intersections of all obstacles with this plane are oriented rectangles. Define w to be a linear combination of y and z directions so that points on this plane can be written in (x, w) coordinates, and translate these so that $s'' = (0, n_w)$, $t' = (n_x, 0)$, and $t = (n_x, n_w)$ in this system.

From t' do a greedy $\langle +w, -x \rangle$ path until either the w-coordinate is n_w or else the x-coordinate is 0, whichever comes first. If the first case occurs, then the greedy path is a monotone boundary and we can apply the room problem algorithm of Theorem 3.2. (The 2-dimensional slice may technically violate our conditions for the room problem by having obstacles that are too "thin". However, because a unit cube can be inscribed in each of the 3-dimensional obstacles, there is sufficient separation for the room-problem algorithm to work.) If the second case occurs, (we reached a point with x-coordinate 0), then go back to s'', retracing all our steps if we have to, and perform a greedy $\langle +x, -w \rangle$ greedy path from there. This is guaranteed to hit a point with x-coordinate of n_x and w-coordinate at least 0 since it cannot cross our previous greedy path. So, we again have a room and can run the room algorithm. The distance walked in this stage is $O(\sqrt{n} \cdot d(\mathcal{S}))$ since $n_w \leq d(\mathcal{S})$. We therefore have the following theorem.

THEOREM 7.4. *For three dimensional scenes $\mathcal{S}$ in which s and t are points and every obstacle is a cuboid whose sides are parallel to the axes our algorithm achieves a ratio of $\rho(R, n) = O(n^{2/3})$.*

To extend this result to a tactile robot we again use the technique due to Baeza-Yates *et al.* [1]. Their strategy allows one to start from a face of a cuboid and travel to the nearest edge (at distance d) walking distance $O(d)$ in the process, without prior knowledge of d or the direction to the nearest edge. As in Section 5, this allows our algorithm for point-to-point navigation in three dimensions to work for tactile robots with the same asymptotic ratio bounds.

8. The Power of Randomization. We now consider randomized robots that toss coins as they walk from s to t. The scene $\mathcal{S}$ is fixed in advance by an *oblivious* adversary [3] who knows the randomized algorithm, but not the coin tosses made by the robot during a walk. The cost of robot R on scene $\mathcal{S}$ is now a random variable; we thus define the ratio $\rho(R,n)$ to be $\sup_{\mathcal{S}\in\mathcal{S}(n)} E[R(\mathcal{S})]/d(\mathcal{S})$. The main result of this section is a randomized algorithm for the 2-dimensional wall problem that achieves a ratio that is $2^{O(\sqrt{\log n \log\log n})}$ provided the obstacles are all vertical line segments with endpoints at integral x-coordinates and the robot is allowed *vision*. Notice that for this situation, the robot can see the entire "column" of obstacles directly in front of it; that is, if the robot is at a point with x-coordinate in the range $(i-1,i)$ for integer i, it can see all obstacles of x-coordinate i. To keep with our previous conventions on the thickness of obstacles, we could equivalently consider obstacles of width between one and two having their left walls only at even x-coordinates; this would still allow the robot to see an entire "column" at once.

The Papadimitriou-Yannakakis lower bound of $\Omega(\sqrt{n})$ still holds for deterministic algorithms for this restricted class of scenes [26]. So, for such scenes, a randomized algorithm is provably better than a deterministic one. We leave as an open question whether one can achieve similar bounds for the more general wall problem.

The idea for the randomized algorithm is to view the problem as a k-server problem on $(k+1)$ equally-spaced points on a line, and then use as a subroutine known randomized strategies [4] for that server problem. For the benefit of the reader, we now define the *k-server problem*, first defined in [23]. An on-line algorithm manages k mobile servers located at the vertices of a graph G whose edges have positive real lengths. The algorithm has to satisfy on-line a sequence of requests, each of which is some vertex v of G, by moving a server to v unless it already has a server there. Each time it moves a server, it pays a cost equal to the distance moved by that server. We compare the cost of such an algorithm to the cost of an adversary that, in addition to moving its servers, also generates the sequence of requests. In fact, our problem can be better described as a *metrical task system* of [6], but we will use the language of servers here. In the lower bound direction, a recent result of Karloff *et al.* for the server problem shows that even for the special case of scenes we consider, no randomized algorithm can achieve a constant ratio [17].

We now present our randomized algorithm. There is a randomized strategy for k servers on $k+1$ equally-spaced points on the line that achieves a competitiveness $2^{O(\sqrt{\log k \log\log k})}$ against the oblivious adversary [4] (for completeness, details are given in the Appendix). We map the navigation problem to this k-server problem as follows. Let $k = n-1$ and define the spacing between adjacent points on the line to be W/n, where W is the width of a window of y-coordinates currently considered by the robot; the value of W will be specified below. Each point in the server problem corresponds to a range of W/n y-coordinates for the navigation problem. The "hole" (the point without a server) represents the range currently inhabited by the robot.

We begin with $W = n$ and start the hole at the center of the line. Each time the robot sees a column of obstacles, the robot notes all points in the server problem corresponding to ranges that are completely blocked by obstacles. It then makes enough requests to the server algorithm on those points so that for the server algorithm of [4], the hole no longer resides on such points. Note that this request sequence is determined by the scene and thus obeys the definition of an oblivious adversary. The robot then moves to the range occupied by the hole (if it is not already there) and then moves a vertical distance at most W/n to find a point where it can go forward in

the $+x$ direction to the next column. So, the distance moved by the robot is at most the on-line server cost, plus $W/n + 1$ for each unit moved in the $+x$ direction. If the off-line server cost reaches W, the robot doubles the window width and restarts the server algorithm, each point now corresponds to a larger range of y values.

THEOREM 8.1. *The randomized algorithm above achieves a ratio that is* $2^{O(\sqrt{\log n \log\log n})}$ *for the wall problem in the plane where the robot uses vision and the obstacles are vertical line segments at integral x-coordinates.*

Proof. For a fixed window width, the off-line server cost in the above transformation is a lower bound on the length of the shortest path for the robot problem (assuming the off-line hole is started also at the center of the line). The off-line server cost could be a bit lower than the length of the shortest path since we do not make requests to points corresponding to y-value ranges only partially blocked by obstacles. Note that when the off-line cost exceeds W, the shortest path might escape the window, which is why W is doubled.

As mentioned previously, the on-line cost for the robot is a most the on-line cost for the server problem, plus $W/n + 1$ for each unit advance in the x direction. So, if W_f is the final window width used, the total distance traveled by the robot is at most $(W_f + n) + W_f 2^{O(\sqrt{\log n \log\log n})} = d(\mathcal{S})2^{O(\sqrt{\log n \log\log n})}$. □

9. Non-convex Obstacles and Mazes. When the obstacles are non-convex, the scene can be a maze. In this case it is easy to see that $\rho(R, n)$ cannot be bounded by any function of n (the Euclidean distance between s and t). Instead, we prove a ratio between $R(\mathcal{S})$ and $d(\mathcal{S})$ as a function of the total number of vertices in all the obstacles, $|V|$.

THEOREM 9.1. *No randomized algorithm achieves a ratio better than* $(|V| - 10)/6$.

Proof. Consider the maze in Fig. 7 and its obvious generalization.

The maze has $(|V| - 10)/6$ passages that could lead from s to t. An algorithm attempts various passages in turn, until it finds the sole passage open to t. For any randomized algorithm, there is one passage whose expected "time to attempt" is at least (Number of passages $- 1)/2$; this passage is left open to t. The robot walks $2d(\mathcal{S})$ on every failure before that attempt, and $d(\mathcal{S})$ on that attempt. □

The bound applies *a fortiori* to deterministic algorithms. Rao *et al.* [28] give a deterministic algorithm that explores a maze by building a map of the scene, proceeding at each step to that unexplored vertex of the maze nearest to the vertices that have already been visited. It is easy to show that this algorithm achieves a ratio of at most $2|V|$, matching the above lower bound to within a constant. This algorithm is memory intensive, and this may be a handicap when space is limited or the scene changes quickly enough that a map is not worth building. We now give a simple, memoryless randomized alternative based on a random walk that works for scenes in the plane. We first define a graph $G(\mathcal{S})$ on the vertices of the polygons in $\mathcal{S}$, and prove a simple geometric property of this graph. We then describe how the robot can perform a random walk on this graph, and invoke a result on random walks to prove that the robot's ratio is $O(|V|)$.

The graph $G(\mathcal{S})$ is defined as follows: each vertex of a polygon in $\mathcal{S}$ is a node in the graph. A node v in $G(\mathcal{S})$ chooses up to twelve neighbors, defined as follows. Consider the twelve cones defined by angular intervals $[\pi i/6, \pi(i+1)/6)$, $i = 1, 2, \ldots, 12$ about v. There is an edge joining v to the nearest visible vertex (if any) in each cone. Thus $G(\mathcal{S})$ has at most $12|V|$ edges. A construction similar to $G(\mathcal{S})$ appears in

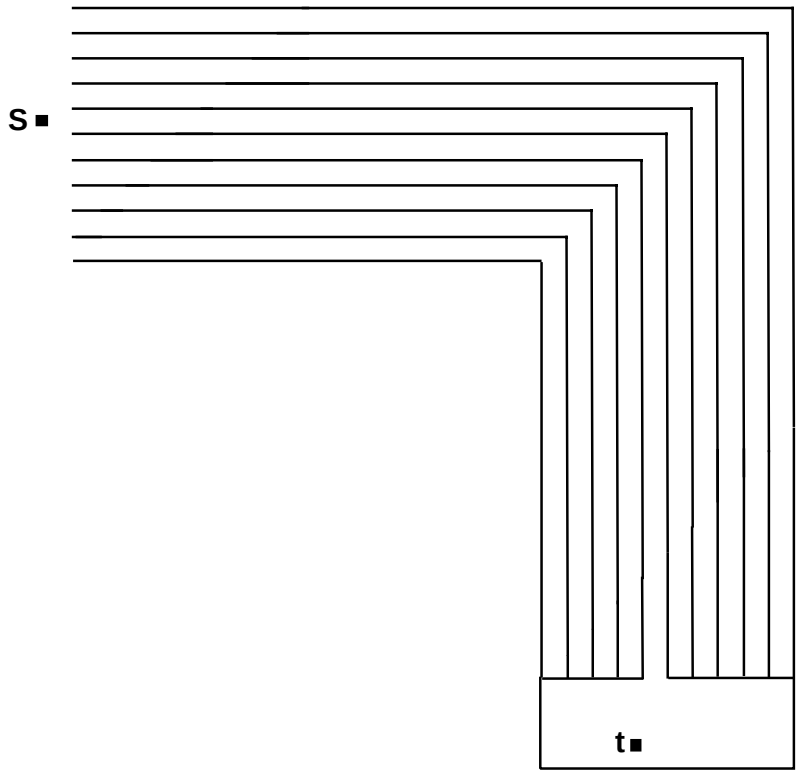

FIG. 7. *A maze achieving the lower bound. Each line segment corresponds to an obstacle with four vertices.*

Clarkson [11], where a result similar to the following lemma was given:

LEMMA 9.2. *Let $d_{st}(\mathcal{S})$ be the distance between two vertices s and t in the scene $\mathcal{S}$. There is a path in $G(\mathcal{S})$ between s and t of length at most $2.1d_{st}(\mathcal{S})$.*

Proof. For two vertices u, v in the scene that are mutually visible, denote by d_{uv} the distance between them. The shortest path in $\mathcal{S}$ between s and t is a path in the *visibility graph* of $\mathcal{S}$ [32]: a graph whose nodes are the vertices of obstacles in $\mathcal{S}$, with two nodes being joined by an edge if they are visible from each other. We now show that given this shortest path (of length $d_{st}(\mathcal{S})$) in the visibility graph, we can find a path in $G(\mathcal{S})$ between s and t whose length is at most $2.1d_{st}(\mathcal{S})$. Note that we can afford to find this path "off-line": we only wish to exhibit the existence of a short path in $G(\mathcal{S})$ from s to t.

We use an iterative strategy: we take the first edge of the visibility graph on the shortest path, say (s, a). If (s, a) is an edge in $G(\mathcal{S})$ we proceed to a and continue from there. Otherwise, we show that there is an edge (s, b) in $G(\mathcal{S})$ with the following property: $d_{bt}(\mathcal{S}) \leq d_{st}(\mathcal{S}) - 0.48d_{sb}$ (note that s and b are mutually visible). We therefore go from s to b in the first step of our path in $G(\mathcal{S})$ from s to t, having ensured that (1) we move to a node whose distance to t in $\mathcal{S}$ is less than from s, and (2) the distance we have walked is proportional to the reduction in the remaining distance. We now continue the iteration from b, with b playing the role of s. In fact, since our distance to t diminishes at each iteration, we will have at most $|V| - 1$ iterations before arriving at t.

It remains for us to bound the first step (s, b) when (s, a) is not an edge in $G(\mathcal{S})$. Since s and a are mutually visible, the only reason that segment sa is not an edge in $G(\mathcal{S})$ is that the cone containing the line segment sa has a node b in it such that (s, b) is an edge in $G(\mathcal{S})$, i.e., $d_{sb} \leq d_{sa}$. Consider the sector of the circle with center s and radius d_{sb} that lies in the cone containing the segment sa. Let c be the point where this sector cuts segment sa. Since b is the closest vertex to s in the cone, no obstacle vertex lies in this sector. Further, since both a and b are visible from s, no portion of any obstacle lies in this sector. Therefore b and c are mutually visible, as are a and c. Fig. 8 illustrates these facts. Thus

$$d_{bt}(\mathcal{S}) \leq d_{bc} + d_{ca} + d_{at}(\mathcal{S}).$$

On the other hand,

$$d_{st}(\mathcal{S}) = d_{sc} + d_{ca} + d_{at}(\mathcal{S}).$$

Using some elementary trigonometry and the fact that the angle between segments sa and sb is at most $\pi/6$, we have

$$d_{bt}(\mathcal{S}) \leq d_{st}(\mathcal{S}) - d_{sc} + d_{bc} \leq d_{st}(\mathcal{S}) - 0.48 d_{sb}.$$

□

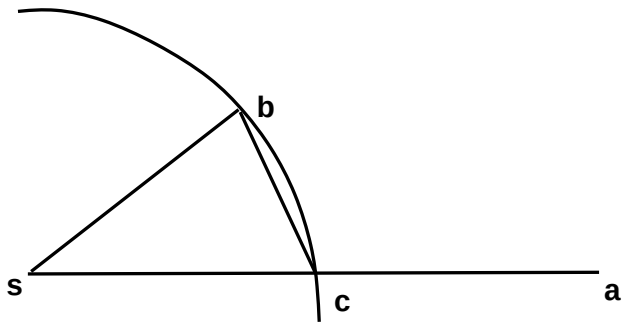

FIG. 8. $d_{bc} < 0.52 d_{sc}$ *and* $d_{sb} = d_{sc}$

We now describe the random walk that the robot executes in going from s to t. At each node, it looks out to see the nearest node in each of the twelve cones, if such exists; let them be $v_1, v_2, \ldots, v_k$, at distances $d_1, d_2, \ldots, d_k$ respectively. Then it chooses to go to v_i with probability

$$\frac{1/d_i}{\sum_{j=1}^{k} 1/d_j}.$$

Note that this probabilistic decision is a local choice that does not need knowledge of $G(\mathcal{S})$ in advance; at each vertex, the robot measures the distance to the nearest visible node in each cone and chooses each with probability inversely proportional to its distance. The robot stops the process on arriving at t.

THEOREM 9.3. *The expected distance traveled by the robot is at most* $50.4|V| d_{st}(\mathcal{S})$. *Thus it achieves a ratio of at most* $50.4|V|$.

Proof. Chandra *et al.* [8] have studied the following general walk in a graph with positive real edge lengths: at each node, the walk chooses the next edge to walk along with probability inversely proportional to its length. They show that the expected distance traversed by the walk in going from a node a to a node b is at most $2m\ell_{ab}$, where m is the number of edges and ℓ_{ab} is the length of the shortest path in the graph between a and b.

In our case, $m \leq 12|V|$, and $\ell_{st} \leq 2.1 d_{st}(\mathcal{S})$ by Lemma 9.2; combining these facts with the result in [8] yields the theorem. □ Clearly there is a tradeoff between the number of cones in the graph $G(\mathcal{S})$ that we define and the factor 2.1 in Lemma 9.2; had we used 36 cones each of angle $\pi/18$, we could have got a tighter factor there but the number of edges in the graph (which figures in the ratio achieved by the random walk) goes up. Our choice of 12 cones optimizes this tradeoff.

10. Open Problems. We conclude with some open problems.

- What are the tight bounds (deterministic as well as randomized) for the room problem with general obstacles?
- Can a randomized algorithm for the room problem beat deterministic algorithms?
- Extend the sweep algorithm for the wall problem to handle arbitrary polygonal obstacles, and hence or otherwise obtain an algorithm for point-to-point navigation with such obstacles.
- Extend all of the above to three dimensions.
- Give an algorithm that achieves a provably good ratio for three-dimensional scenes with non-convex obstacles (three-dimensional mazes).
- Blum and Kozen [5] show that a planar maze can be traversed in a number of steps polynomial in the number of vertices in the maze, by a deterministic automaton using two pebbles. We have seen that the deterministic algorithm of Rao *et al.* achieves an optimal ratio but is memory-intensive, whereas the random walk achieves a similar ratio without using memory to build a map. Is there a deterministic automaton using few pebbles (small memory) that achieves a good ratio? It seems reasonable to expect that the automaton would need a distance counter as well.

Acknowledgements. We thank Alok Aggarwal, Allan Borodin, Don Coppersmith, Leo Guibas, Sandy Irani, Ming Kao, Howard Karloff, Samir Khuller and Yishay Mansour for comments and suggestions. We also thank the referees for many valuable comments.

Appendix. We outline the randomized k-server algorithm invoked in Section 8, and the proof of its competitiveness.

THEOREM 10.1 ([4]). *There is a randomized algorithm for $k = n - 1$ servers on n equally-spaced points on a line that achieves an expected competitiveness ratio of $2^{O(\sqrt{\log n \log\log n})}$ against an oblivious adversary.*

Proof: Without loss of generality assume that the points are spaced at unit distance. For convenience, we call the point without any server the "hole" and think of the algorithm as being on the hole position, and having to move when it is "hit" by a request. The idea of the algorithm is to break up the line into a collection of equal-sized intervals and then to stay within some interval until the adversary has made "enough" requests inside it. Once the adversary has made enough requests, the algorithm moves the hole to a different interval, choosing at random from those intervals into which "not too many" requests have been made.

More specifically, the algorithm proceeds as follows. Let $m = \left\lceil n/2^{\sqrt{\log n \log\log n}} \right\rceil$.

Algorithm **Randomized-Line**

Step 1: Break the line into $\lceil n/m \rceil$ intervals, each of m points except possibly the last. We label these intervals $I_1, \ldots, I_{\lceil n/m \rceil}$. Initialize each interval to be "unmarked."

Step 2: For each point i $(1 \leq i \leq n)$, initialize a variable $C(i)$ to zero. Each $C(i)$ represents the minimum possible off-line cost of ending at point i given the sequence of requests seen since the last initialization (and assuming the off-line server may start at any point). Updating $C(i)$ is easy: after a request is made at point i, $C(i) \leftarrow \min\{C(i-1)+1, C(i+1)+1\}$. To handle the endpoints, initialize $C(0)$ and $C(n+1)$ to infinity.

Step 3: Randomly choose one unmarked interval I_j. Stay inside the larger interval $I_{j-1} \cup I_j \cup I_{j+1}$ running Randomized-Line recursively within that region, until the minimum cost $C(i)$ for $i \in I_j$ has risen to be greater than $m/2$. As a base case, for a small enough interval, any deterministic algorithm will do (e.g., the deterministic algorithm given in [10]). For consistency at the endpoints, define I_0 and $I_{\lceil n/m \rceil + 1}$ to be empty and always marked.

Remark: The reason for staying within a larger interval of size $3m$ is a technical one to handle the "edge effects" that occur at the boundaries of the intervals I_j, as discussed in the analysis.

Step 4: Mark all intervals I_j such that the minimum cost $C(i)$ for all $i \in I_j$ is at least $m/2$. If there is some unmarked interval left, then go back to Step 3.

Step 5: All intervals are now marked, so the off-line cost since the last initialization of the $C(i)$'s is at least $m/2$. Go back to Step 2 and reinitialize.

Analysis: Let $T(n)$ be the expected cost of algorithm Randomized-Line for $n-1$ servers on a line of n unit-spaced points, for a sequence of requests yielding a minimum off-line cost of $\lceil n/6 \rceil$.

Each application of Step 3 costs the algorithm at most an expected $n + T(3m)$; n for moving to the chosen interval and $T(3m)$ for the cost inside that interval. Notice that the cost function $C(i)$, for $i \in I_{j-1} \cup I_j \cup I_{j+1}$, may not yield values as high as those computed by the recursive application. The reason is the "edge effects": in the recursive application the off-line costs of the endpoints of the range are not constrained by the costs of points outside it. However, the range is large enough so that *for all i in the middle region I_j*, the cost $C(i)$ is at least that computed by the recursive application as long as $C(i) \leq \lceil m/2 \rceil$.

Each application of Step 3 results in half of the unmarked intervals becoming marked in Step 4 on average, since the central interval I_j inhabited by the hole is chosen randomly from the unmarked intervals. So, after $O(\log n)$ applications of Step 3 with high probability all intervals have been marked. Once we repeat $\lceil n/(3m) \rceil$ times steps $2-5$, the off-line cost has increased by at least $n/6$.

So, we get the following recurrence:

$$\begin{aligned} T(n) &\leq \lceil n/3m \rceil \left[O(\log n)[T(3m)+n] \right], \quad \text{for, say, } n \geq 16 \\ &\leq \frac{c(n \log n)}{m} \left[T(3m) + n \right], \quad \text{for some constant } c. \end{aligned}$$

Substituting $m = \left\lceil n/2^{\sqrt{\log n \log\log n}} \right\rceil$ yields

$$T(n) \;=\; O(n \cdot 2^{3\sqrt{\log n \log\log n}}). \quad \square$$

REFERENCES

[1] R.A. Baeza-Yates, J.C. Culberson, and G.J.E. Rawlins, *Searching in the plane*, Information and Computation, vol. 106, no. 2, pp. 234–252, October 1993.

[2] E. Bar-Eli, P. Berman, A. Fiat, and P. Yan, *On-line navigation in a room*, Journal of Algorithms, 17:319–341, 1994.

[3] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson, *On the power of randomization in on-line algorithms*, Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, pp. 379–388, 1990.

[4] A. Blum, A. Borodin, D. Foster, H.J. Karloff, Y. Mansour, P. Raghavan, M. Saks, and B. Schieber, *Randomized on-line algorithms for graph closures*, Personal communication, 1990.

[5] M. Blum and D. Kozen, *On the power of the compass (or, why mazes are easier to search than graphs)*, Proceedings of the 19th Annual Symposium on Foundations of Computer Science, pp. 132–142, October 1978.

[6] A. Borodin, N. Linial, and M. Saks, *An optimal online algorithm for metrical task systems*, Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, pp. 373–382, 1987.

[7] S.J. Buckley, *Planning compliant motion strategies*, International Journal of Robotics Research, 8:28–44, October 1989.

[8] A. K. Chandra, P. Raghavan, W.L. Ruzzo, R. Smolensky, and P. Tiwari, *The electrical resistance of a graph captures its commute and cover times*, Proceedings of the 21st Annual ACM Symposium on Theory of Computing, pp. 574–586, Seattle, May 1989.

[9] L. Cheng and J.D. McKendrick, *Autonomous knowledge based navigation in an unknown two dimensional environment with convex polygonal obstacles*, Proceedings of the Int. Soc. Opt. Eng., volume 1095, pp. 752–759, 1989.

[10] M. Chrobak, H.J. Karloff, T. Payne, and S. Vishwanathan, *New results on server problems*, Proceedings of the First ACM-SIAM Symposium on Discrete Algorithms, pp. 291–300, 1990.

[11] K.L. Clarkson, *Approximation algorithms for shortest path motion planning*, Proceedings of the 19th ACM Symposium on Theory of Computing, pp. 56–65, May 1987.

[12] E.G. Coffman and E.N. Gilbert, *Paths through a maze of rectangles*, Networks, vol. 22, no. 4, pp. 349–367, July 1992.

[13] M. Daily, J. Harris, D. Keirsey, D. Olin, D. Payton, K. Reiser, J. Rosenblatt, D. Tseng, and V. Wong, *Autonomous cross-country navigation with the ALV*, Proceedings of the IEEE International Conference on Robotics and Automation, volume 2, pp. 718–726, 1988.

[14] P. Eades, X. Lin, and N.C. Wormald, *Performance guarantees for motion planning with temporal uncertainty*, Aust. Comput. Journal, 25(1):21–28, February 1993.

[15] J. Hallam, P. Forster, and J. Howe, *Map free localization in a partially moving 3-D world: the Edinburgh feature based navigator*, Proc. Intl. Conf. Intelligent Autonomous Systems, volume 2, pp. 726–736, 1989.

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

[17] H.J. Karloff, Y. Rabani, and Y. Ravid, *Lower bounds for randomized server algorithms*, Proc. 23rd ACM Symposium on Theory of Computing, pp. 278–288, 1991.

[18] R. Klein, *Walking an unknown street with bounded detour*, 32nd Annual IEEE Symposium on Foundations of Computer Science, pp. 303–313, 1991.

[19] V. Lumelsky, *Algorithmic issues of sensor-based robot motion planning*, 26th IEEE Conference on Decision and Control, pp. 1796–1801, 1987.

[20] ——*Algorithmic and complexity issues of robot motion in an uncertain environment*, Journal of Complexity, 3:146–182, 1987.

[21] V.J. Lumelsky and A.A. Stepanov, *Dynamic path planning for a mobile automaton with limited information on the environment*, IEEE Transactions on Automatic Control, AC-31:1058–1063, 1986.

[22] A. Mei and Y. Igarashi, *An efficient strategy for robot navigation in unknown environment*, Information Processing Letters, 52:51–56, 1994.

[23] M.S. Manasse, L.A. McGeoch, and D.D. Sleator, *Competitive algorithms for on-line problems*, Journal of Algorithms, 11:208–230, 1990.

[24] H. Moravec, *The Stanford cart and the CMU rover*, Proceedings of the IEEE, 71:872–874, 1983.

[25] B.J. Oomen, S.S. Iyengar, N.S.V., Rao, and R.L. Kashyap, *Robot navigation in unknown terrains using learned visibility graphs. Part I: The disjoint convex obstacle case*, IEEE Journal of Robotics and Automation, 3:672–681, 1987.

[26] C.H. Papadimitriou and M. Yannakakis, *Shortest paths without a map*, Proc. 16th ICALP, pp. 610–620, July 1989.

[27] N.S.V. Rao, *Algorithmic framework for learned robot navigation in unknown terrains*, IEEE COMPUTER, 22:37–43, 1989.

[28] N.S.V. Rao, S.S. Iyengar, and G. deSaussure, *The visit problem: visibility graph based solution*, IEEE International Conference on Robotics and Automation, pp. 1650–1655, 1988.

[29] N.S.V. Rao, S.S. Iyengar, B.J. Oomen, and R.L. Kashyap, *On terrain model acquisition by a point robot amid polyhedral obstacles*, IEEE Journal of Robotics and Automation, 4:450–455, 1988.

[30] C.N. Shen and G. Nagy, *Autonomous navigation to provide long distance surface traverses for Mars rover sample return mission*, IEEE International Symposium on Intelligent Control, pp. 362–367, 1989.

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

[32] C-K. Yap, *Algorithmic motion planning*, In J.T. Schwartz and C.K. Yap, editors, *Advances in Robotics*, pp. 95–144. Lawrence Erlbaum Associated, Hillsdale, NJ, 1987.