Chapter 1

Randomized Robot Navigation Algorithms

Piotr Berman [*] Avrim Blum [†] Amos Fiat [‡] Howard Karloff [§] Adi Rosén [¶] Michael Saks [||]

Abstract

We consider the problem faced by a mobile robot that has to reach a given target by traveling through an unmapped region in the plane containing oriented rectangular obstacles. We assume the robot has no prior knowledge about the positions or sizes of the obstacles, and acquires such knowledge only when obstacles are encountered. Our goal is to minimize the distance the robot must travel, using the competitive ratio as our measure.

We give a new randomized algorithm for this problem whose competitive ratio is $O(n^{\frac{4}{9}} \log n)$, beating the deterministic $\Omega(\sqrt{n})$ lower bound of [PY], and answering in the affirmative an open question of [BRS] (which presented an optimal deterministic algorithm). We believe the techniques introduced here may prove useful in other on-line situations in which information gathering is part of the on-line process.

1 Introduction

Robotics applications often require that a robot move through an obstacle-filled region to reach a specified target. Usually the robot is given freedom as to what route to follow, and we wish to minimize the distance traveled. Two very different problems arise from these situations, depending on the prior knowledge the robot has about the region. If the robot knows the positions of the obstacles in the region, the problem is a computational question of computing a short, sometimes the shortest, path avoiding the obstacles to reach the target (see [La, Sh]). Such situations may arise if the robot is traveling on a factory floor that bears no changes over time, or if we have a map of the region to be traversed.

A very different problem arises when the robot has no *a priori* information about the obstacles that lie ahead (for instance, if the factory floor changes frequently, or the robot is placed in a completely new region). We model this situation by assuming that the robot knows the location of the target and its own absolute position, but that it only acquires knowledge about the obstacles as it contacts them. Our aim, nonetheless, is to minimize the distance the robot travels.

In recent years, several theoretical papers have studied problems of this second type in which the robot has no prior knowledge of the positions of the obstacles [PY, BRS, BC, BBFY]. Because the situation examined is "on-line" in nature, in the sense that the robot must make decisions without knowing what lies ahead, it is natural to use the *competitive ratio* [ST] to evaluate the performance of a strategy. In particular, we would like to minimize the ratio between the distance traveled by the robot and the length of the shortest start-to-target path in that scene. The competitive ratio is the worst case ratio achieved over all scenes having a given Euclidean source-target distance.

A particularly interesting scenario, that embodies many of the key issues that arise, is the case that all obstacles are axis-parallel rectangles. In fact, it is often convenient (and nearly equivalent [BRS, BBFY]) to further simplify and assume that the goal is just to advance some number of units forward in the x direction (i.e., the target is an infinite vertical "wall"). For either version, it is not hard to see that the simple strat-

[*] Computer Science Dept., Penn State University, University Park, PA 16802. berman@shire.cs.psu.edu

[†] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891. Supported in part by NSF National Young Investigator grant CCR-9357793 and a Sloan Foundation Research Fellowship. avrim@cs.cmu.edu

[‡] Dept. of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. fiat@math.tau.ac.il

[§] College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280. Research supported in part by NSF grant CCR-9319106. howard@cc.gatech.edu

[¶] Dept. of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. adiro@math.tau.ac.il

[||] Department of Mathematics, Rutgers University, New Brunswick, NJ 08903. Supported in part by NSF contract CCR-9215293 and by DIMACS, which is funded through NSF grant STC–91–19999 and the NJ Commission on Science and Technology. saks@dimacs.rutgers.edu

egy of greedily moving towards the target and going greedily around any obstacles encountered has competitive ratio $\Theta(n)$, where n is the Euclidean source-target distance and we define our units so that all obstacles have length and width at least one. Similar greedy strategies fare equally poorly. Blum, Raghavan, and Schieber [BRS], however, gave a more clever deterministic algorithm that achieves a competitive ratio of $O(\sqrt{n})$ for both the "wall" and point-to-point problems. This matches an $\Omega(\sqrt{n})$ lower bound that had been proven by Papadimitriou and Yannakakis [PY], closing the gap for deterministic algorithms. [BRS] posed the question of whether *randomized* algorithms might be able to do better.

In this paper, we break the deterministic $\Theta(\sqrt{n})$ barrier by describing a new randomized navigation algorithm whose competitive ratio is $O(n^{\frac{4}{9}} \log n)$, thus answering the open question of [BRS] in the affirmative. We also show (details in the full paper) that randomization provably helps in the 3-dimensional version of the problem, giving an $O(n^{2/3-\epsilon})$-competitive algorithm.

While the new ratio admittedly provides only a small reduction in the exponent over the deterministic results, we believe the robot navigation problem is particularly intriguing from the point of view of randomization for the following reason. Unlike on-line scenarios such as the k-server [MMS] or metrical task system [BLS] problems, in the robot navigation setting one has missing information about the *past* (obstacles closer to the start than the robot's current position but that were not actually touched by the robot) as well as the future. The path that one takes determines the information that one receives. What this means is that one cannot easily determine the optimal cost from the start to other points at the current "level" (x coordinate), which is something used by standard randomized on-line algorithms [FKLMSY]. This is somewhat like a paging problem in which the algorithm is only told about the page faults and not the "hits". In fact, for that version of the paging problem it has been shown that randomization provably does *not* help [Ch].[1] We believe that our methods are the first in the framework of competitive analysis to show how randomization can be usefully employed when information gathering is part of the on-line process.

The best lower bound known for randomized navigation algorithms is $\Omega(\log \log n)$ [KRR], and this lower bound is based on the simpler case in which positions of all obstacles with x-coordinate less than that of the robot are revealed as the robot travels. An obvious open question is whether the best competitive ratio achievable is something like $O(\log n)$ as is suspected to be the correct answer for that simpler case, or whether there is an $\Omega(n^c)$ lower bound.

To provide intuition for some of the difficulties in using randomization for navigation, in the appendix we prove that any randomized strategy that fails to explicitly keep track of its x-motion (the deterministic strategy of [BRS] falls into this category) has an $\Omega(\sqrt{n})$ lower bound.

1.1 Related Theoretical Work

In addition to theoretical work on robot navigation in unknown environments, mentioned above, several additional papers on robot motion planning use the measure of competitive ratio to evaluate performance. The problems addressed include *search* problems, where a robot has to find some entity in an unknown location [BCR, KRT, Kl], and *exploration* problems, where a robot has to gain full information of the region, by going along a path from which it can observe all the boundaries defining the region [DKP, Kl].

2 Preliminaries

2.1 The Model

We consider the setting of a planar region containing axis-parallel rectangular obstacles. All sides of the obstacles have length at least 1 and the obstacles do not overlap. We treat the robot as a point particle that cannot pass through obstacles but may squeeze between two adjacent ones. Thus one cannot "combine" abutting rectangles to make a more complicated obstacle. We refer to a legal path through the region as an *obstacle-free* path.

For the *wall problem*, the robot has to travel from its starting point $s = (0, 0)$ to a point of its own choosing having x-coordinate equal to n (i.e., to a point on an infinite vertical "wall" at $x = n$). For *point-to-point navigation*, the robot has to reach one given point t, rather than a point of its choosing on the infinite wall. The point t must not, of course, reside in the interior of any of the obstacles.

We assume that the robot has no prior knowledge of the positions or sizes of any of the obstacles and acquires knowledge about an obstacle only when it

[1] The paging result holds even when there is exactly one more page than slots in fast memory. This can be viewed as a navigation-like problem by thinking of each page as representing a y-coordinate, and the page not in fast memory as the y-coordinate of the robot. A request to a page corresponds to an obstacle at that y-coordinate. Key differences, however, are (1) the paging problem has a uniform, not linear, cost metric, (2) the paging problem is assumed to go on indefinitely whereas the navigation problem has a fixed ending point at $x = n$, and (3) in the navigation problem one can "back up" if desired, though our algorithm does not do so.

hits the obstacle (a *tactile* robot). We further assume that when the robot hits an obstacle it also knows the direction, up or down, to the nearest corner. Using the techniques of Baeza-Yates, Culberson, and Rawlins [BCR] this last assumption can be eliminated by adding only a multiplicative constant factor to the cost of the algorithm.

A *scene* S is an obstacle-filled region together with a target wall or point. We parameterize the scenes by the *Euclidean* distance between the start-point and the target. The set $\mathcal{S}(n)$ is the set of scenes in which this distance is $n \geq 1$. Given a strategy R for the robot and a scene S, we denote by $R(S)$ the distance traveled by the robot until it reaches the target in scene S. When the strategy is randomized, $R(S)$ is a random variable. We define a path to be *Manhattan* if it consists only of line segments parallel to either of the coordinate axes. We denote by $\mathrm{OPT}(S)$ the length of the shortest Manhattan obstacle-free path from the starting point to the target in the scene S. It is not hard to see that restricting the shortest path to be Manhattan increases its length by only a constant factor.

The competitive ratio [ST] of a strategy R for $\mathcal{S}(n)$ is

$$C(n) = \sup_{S \in \mathcal{S}(n)} \frac{E[R(S)]}{\mathrm{OPT}(S)} \ ;$$

the expectation is over the random choices made by the algorithm during its execution. We wish to minimize this ratio (as a function of n).

When the on-line algorithm is randomized, it is important to specify the power of the adversary against which it competes [BBKTW]. The above definition implies an *oblivious* adversary, which has to commit to a scene before the on-line strategy starts to move the robot. In other words, the adversary knows the strategy that controls the robot, but cannot see the random choices it takes.

Throughout this paper, we refer to the x direction as "horizontal" and the y direction as "vertical". We also use the terms *right*, *left*, *up*, and *down* to refer to the positive-x, negative-x, positive-y, and negative-y directions respectively. Finally, we use *width* to refer to length in the horizontal direction and *height* to refer to length in the vertical direction.

2.2 Sweeps

We use in our algorithms a notion of a *sweep* [BRS] (also called a *fence* [BC]).

Let $a < b$. A *corridor* is the region $\{(x, y) | a \leq y \leq b\}$, and it is said to be *defined by* $[a, b]$.

A (y_1, y_2)-*sweep with threshold* τ is the following simple strategy to be employed by a robot currently at a point with y-coordinate y_1. Let C be the corridor defined by $[y_1, y_2]$ (so the robot is at the bottom edge of C). At a generic time during the sweep, the robot is either freely moving rightward (horizontally), or, after colliding with an obstacle during its rightward horizontal motion, it is moving around the obstacle. When a collision occurs, the robot considers the distance to the nearest corner. If the distance to the nearest corner is at most τ, then the robot "goes around" the obstacle, first moving to the nearest corner, then moving horizontally to the right edge of the obstacle, and then moving up or down back to a point with the same y-coordinate as the point of impact. (This happens even if the robot temporarily has to leave the corridor C.) The total vertical distance traveled in going around the obstacle in this case is at most 2τ. Otherwise (the distance to the nearest corner exceeds τ), the robot just travels upward until it reaches the upper-left corner of the obstacle, at which point it reverts to moving horizontally rightward. This process continues until an instance of the second case occurs which would involve going to or past the line $y = y_2$. At this point, the robot stops at the line $y = y_2$ (the top of C), and starts a version of the same process, but this time moving *downward* in the second case instead of upward. That is, it "goes around" obstacles whose impact point is within τ of the nearest corner, and goes downward when it hits other obstacles, until it hits the bottom edge $y = y_1$ of the corridor. We call this entire procedure one *full sweep*. Notice that if there is a single obstacle extending across the entire corridor then even a full sweep may make horizontal progress 0. The crucial fact about a full sweep is:

Claim 2.1. *Suppose a full sweep begins at (x_1, y_1) and ends at (x_2, y_1). Suppose also there is no obstacle which both extends across the entire corridor and whose left edge is on the line $x = x_2$. Then, any path P beginning at x-coordinate x_1 and ending at x-coordinate x_2 and lying entirely within C must have vertical cost at least τ in the region $x_1 < x < x_2$.*

When the corridor in which the sweep is performed is clear, we sometimes abuse notation and do not indicate it.

When the robot, located in C, wants to perform sweeps in corridor C, it may not be at the bottom of C. In this case, it first moves greedily rightward, moving downward whenever touching an obstacle, until it reaches the boundary $y = y_1$.

3 The Algorithm

In this section we describe our randomized algorithm for the wall problem: i.e., the problem of advancing a given distance n in the positive x direction. In Section 4 we show how this algorithm is easily converted to a point-to-point navigation algorithm with the same competitive ratio, using the techniques of [BRS].

3.1 The high level

The algorithm described below has the property that at all times, the robot either moves horizontally in the positive x-direction, or vertically. Thus, its cost along the x-axis is exactly n. It what follows we refer by *cost* to the distance moved vertically. We prove our competitive ratio with respect to this cost, both for our algorithm and the optimum path. Clearly this competitive ratio is an upper bound on the competitive ratio when both horizontal and vertical motion are considered, as both our algorithm and the optimal path have cost exactly n for horizontal motion.

Our algorithm will run in *stages*. In stage i we maintain a *window* of height $2W_i$, which includes all points with y-coordinate between $-W_i$ and W_i. In each stage the robot moves only within the window. The W_i's satisfy $W_i = 2W_{i-1}$ for all $i > 1$, so $W_i = 2^{i-1}W_1$. To define W_1, the algorithm moves rightward from the origin until it hits the first obstacle. W_1 is twice the distance from the point of impact to the nearest corner of that obstacle. We continue to run new stages, with progressively larger and larger windows, until we reach the wall. Let OPT be the cost incurred for vertical motion by the shortest (Manhattan) path.

In the next sections we will describe the interesting parts of the algorithm (namely, what it does during a stage) and prove the following key properties of the algorithm.

1. The cost in stage i is $O(W_i \cdot n^{\frac{4}{9}} \log n)$.
2. For large enough n, and for each i such that $W_i >$ OPT, the probability that the robot reaches the wall by the end of stage i is at least $3/4$.

Based on the above two key properties it is easy to prove the following theorem.

THEOREM 3.1. *The Algorithm is $O(n^{\frac{4}{9}} \log n)$-competitive.*

Proof. Define $W_0 = W_1/2$. Note that $W_0 \leq$ OPT by the definition of W_1. Let j be such that $W_{j-1} \leq \text{OPT} < W_j$. Therefore the expected cost of the algorithm is at most a constant times

$$\sum_{i=1}^{j-1} W_i \cdot n^{\frac{4}{9}} \lg n + \sum_{i=j}^{\infty} (1/4)^{i-j} W_i \cdot n^{\frac{4}{9}} \lg n.$$

Using

$$\sum_{i=1}^{j-1} W_i < 2W_{j-1} \quad \text{and} \quad \sum_{i=j}^{\infty} (1/4)^{i-j} W_i < 2W_{j-1},$$

it follows easily that

$$\begin{aligned} \sum_{i=1}^{j-1} W_i \cdot n^{\frac{4}{9}} \lg n + \sum_{i=j}^{\infty} (1/4)^{i-j} W_i \cdot n^{\frac{4}{9}} \lg n & \\ < \; & 4W_{j-1}(n^{4/9} \lg n) \\ \leq \; & (4n^{4/9} \lg n)\text{OPT}. \end{aligned}$$

□

3.2 A Stage

3.2.1 Overview and intuition

We will think of the algorithm as competing against an adversary who takes the shortest path. The first idea of the algorithm is to divide the window into corridors of some height h. The algorithm's goal is to either force all corridors to be expensive, in the sense that if the adversary starts in some corridor i then it is expensive for it to remain inside it, or else find some cheap corridor(s) to stay in. A good notion of "expensive" is cost $\Omega(h)$, since at that point it becomes cheaper for the adversary to leave its corridor than to stay inside. In fact, by defining the corridors using a random offset, there is a reasonable chance that the adversary starts near the middle of some corridor, and so having all corridors be expensive provides a lower bound on the adversary's cost.

The standard deterministic way to force a corridor to be expensive is to perform a sweep inside it. But, this can be expensive for the algorithm too. So, in order to more cheaply estimate a corridor's cost, we would like instead to sample it at a *random* x-interval. To do this without incurring too much cost ourselves, we can pick a random starting corridor and then visit corridors in consecutive order, with wrap-around, until we return to our original one. At first glance, one might try to do this by staying inside each corridor until a given amount of effort is spent, and then going on to the next corridor. The problem with this approach, however, is that the adversary can "funnel" the algorithm into a known location by appropriately placing obstacles. Therefore, instead we want to visit each corridor for a fixed x-distance, which we call a *field*, before going on to the next one. This ensures that we visit each corridor at a random field.

Some technical difficulties with this general plan are that if a corridor is *very* expensive we might have to leave it before we are done with the field; or if it happens again, we might have to leave a corridor before we have even started. These issues are dealt with in the details of the algorithm.

At the end of our round trip through the corridors, some corridors will be marked as expensive and some not. We then repeat the process for the cheap corridors. As the number of cheap corridors decreases, in order to pay for our fixed cost of visiting all of them (there is no guarantee the cheap corridors are near to each other), we will make the field widths correspondingly longer. We continue this process until all corridors are marked expensive.

When the process completes we have shown that the adversary had to pay cost $\Omega(h)$ (if it started near the middle of some corridor). We want a bound of $\Omega(W_i)$, so we just repeat the process (with new random offsets) $\Theta(W_i/h)$ times.

In our terminology, one round trip through the corridors is a "subphase", and a sequence of subphases until all corridors are marked expensive is a "phase", which is repeated $\Theta(W_i/h)$ times in a stage.

3.2.2 Details

We now formally define the algorithm in stage i. We make use of a few positive parameters L, α, and r. Optimizing inequalities that result from the algorithm yields $L = n^{\frac{4}{9}}$, $\alpha = n^{\frac{2}{3}}$, and $r = n^{-\frac{1}{9}}$.

Recall that in stage i the algorithm maintains a window $\{(x,y) | -W_i \leq y \leq W_i\}$, and the robot moves only within this window. Define $h = W_i/L^{\frac{1}{2}}$ (this is the corridor height discussed above). A stage consists of $\lceil 32W_i/h \rceil$ *phases* which in turn are divided into *subphases*.

At the beginning of a phase we choose a real *vertical offset* O in the interval $[0,h]$ uniformly at random. We then define at most $2\lceil W_i/h \rceil + 1$ corridors using this offset, all of which (except possibly the two extreme ones) have height h. The corridors are $[-W_i, -W_i+O]$, $[-W_i+O, -W_i+O+h]$, $[-W_i+O+h, -W_i+O+2h]$, and so on, until the last corridor whose upper boundary is at W_i.

At all times, a corridor is in exactly one of three states: *active*, *inactive*, or *useless*. We use the term *non-active* to refer to corridors which are either inactive or useless. ("Non-active" is the negation of "active"; "inactive" is not.) Intuitively, active corridors are cheap and inactive and useless corridors are those that have been found to be expensive in different ways.

At the beginning of a phase all corridors are active. During the execution of a subphase some corridors may be marked inactive and some may be marked as useless. Unless we reach the wall, we continue to run subphases of the same phase, as long as at least one corridor is active. Once all corridors have become non-active (either inactive or useless), we start a new phase with all corridors active again.

A subphase is defined over a vertical strip of width α (an x-distance α). Denote by a_j, $1 \leq a_j \leq 2\lceil W_i/h \rceil + 1$, the number of corridors that are active at the beginning of the jth subphase. We partition the vertical strip of width α into a_j disjoint vertical strips of equal width. This splits each corridor into a_j *fields*. (A field, unlike a corridor, is a (bounded) rectangle. It has bounded width and height, while a corridor has infinite width.) If we look only at the a_j active corridors at the beginning of a subphase (they may not be contiguous, of course), then the fields of the active corridors form an $a_j \times a_j$ checkerboard. We choose a corridor C among the a_j active corridors uniformly at random to be the first corridor. Starting from the leftmost field in that random corridor, we define a diagonal by considering the second field in the next highest active corridor D above C, the third field in the next highest active corridor E above D, etc., wrapping around to the lowest active corridor when there are no more active corridors above. This defines a "diagonal" in the $a_j \times a_j$ checkerboard of fields in the active corridors. (Of course, most or all of a field may lie in one big obstacle.) Our algorithm's goal is to move along this diagonal to get a "random" sample of all the active corridors.

We now define how the robot behaves during a subphase. At all times in a subphase, except during the "fill-up" process at the end described below, the robot has in mind a *desired field* F which is one of the fields on the diagonal. At any point in time the robot will be either: (1) inside the desired field doing sweeps with a small threshold, or (2) inside the corridor containing the desired field, but to the left of the field, doing sweeps with a much larger threshold, or (3) attempting to move between corridors towards the corridor containing the desired field. When we are within the desired field we perform sweeps with threshold $\tau_1 = r \cdot (W_i/L)$ within its corridor. If we are in the corridor to the left of the desired field, moving towards the field, we perform sweeps within the corridor with threshold $\tau_2 = W_i/L$. We will call the sweeps outside and to the left of the desired field *big-τ* sweeps, since their threshold τ_2 is much larger than τ_1, the threshold for the *little-τ* sweeps in the desired fields.

When the subphase starts, the desired field is the initial field on the diagonal. To reach a desired field, the algorithm performs a *find-field* procedure: It takes a greedy path in which it moves right if possible and moves either up or down (whichever brings it closer to the corridor in which the desired field lies) if it cannot move right. If it so happens that the robot's x-coordinate passes the right edge of the desired field before its corridor is reached, then the robot simply updates the desired field to be the next one on the diagonal and continues the procedure. Otherwise, the robot stops when it enters the corridor of its desired field.

After the corridor C of the desired field is reached, then as described above there are two main cases that determine the robot's behavior. The first case is that the robot is actually in the desired field F. In this case, the robot greedily moves to the bottom of C and starts τ_1-sweeps (little-τ sweeps) within F, until either $\lceil \frac{1}{r} \cdot (L^{\frac{1}{2}}/a_j) \rceil$ full little-τ sweeps have been completed in the field, or else the right side of the field has been reached. In the former case we declare the current corridor to be *inactive*. In either case, once we have finished, we run the *find-field* procedure to get to the (corridor of the) next field in our list.

The second case is that the robot is in the corridor C of the desired field F, but to the left of the start of F. In this case the robot greedily moves to the bottom of C and starts τ_2-sweeps (big-τ sweeps) within the corridor, until either the left end of F is reached, or we have completed $L^{\frac{1}{2}}$ full big-τ sweeps in that corridor *counted over the whole phase* (not just the subphase). In the latter case we declare C to be *useless*, and then run the find-field procedure to reach the next desired field. Note that by properties of the sweep, a corridor is useless only if the cost of any path lying entirely within the corridor is at least $W_i/L^{\frac{1}{2}} = h$.

One final case is that the robot has no more "desired fields" but does not reach the right edge of the subphase strip. This can happen if the last field F on the diagonal is marked either inactive or useless before its right edge is encountered (which is the same as the right edge of the subphase). This is a separate case because there is no "next field" to go to. We call this the "fill-up" case. What the robot does now is this: it greedily moves up (as in find-field) to the nearest non-useless corridor—that corridor may either be active or inactive (note that it could be the corridor in which the robot already is)—and performs τ_2-sweeps in this corridor until either the right edge of the subphase is reached or else the corridor can be marked as useless. In the latter case we again greedily move upwards (with wrap-around) to the next non-useless corridor and repeat this procedure. If all corridors are marked useless before the right edge of the subphase is reached, then the phase ends.

3.3 Analysis of a Stage

We start with a technical claim that we later use.

CLAIM 3.1. *Let $a_1 \geq a_2 \geq \ldots \geq a_{l-1} > 0$ be* integers. *Define $a_l = 0$. Then $\sum_{j=1}^{l-1}(a_j - a_{j+1})/a_j \leq 1 + \ln a_1$.*

The simple proof is omitted.

We first prove the following lemma. Note that this bound is a *deterministic* bound.

LEMMA 3.2. *The cost of the algorithm in stage i is $O(W_i \cdot n^{\frac{4}{9}} \log n)$.*

Proof. We analyze below the cost of the algorithm along the y direction. We partition it into several parts.

1. The cost incurred while performing procedure *find-field* is $O(W_i)$, for each subphase. This covers also the greedy moving costs for the "fill-up" case, and it also includes the cost, for each corridor, of moving to the bottom of the corridor to start sweeping (with either threshold, either before fill-up or during it).

   All subphases cover a distance α along the positive x direction, except for possibly the last one in a phase. Thus, there are at most n/α subphases of full-length, and at most $32L^{\frac{1}{2}}$ additional "truncated" subphases (because there are at most $32L^{\frac{1}{2}}$ phases in a stage). Clearly, in one stage the robot can cover at most a distance of n along the x direction; therefore, the total of these costs, over all subphases in the stage, is $O(W_i \cdot (n/\alpha + L^{\frac{1}{2}}))$, which is $o(W_i \cdot n^{\frac{4}{9}} \log n)$.

2. Next we consider the cost incurred while performing little-τ sweeps in a field in the case that we do *not* mark its corridor inactive. Consider subphase j of a certain phase. Denote by a_j the number of corridors that are active at the beginning of that subphase. Each field in that subphase has width α/a_j. The cost in the field is $O((\alpha/a_j)\frac{rW_i}{L} + \lceil L^{\frac{1}{2}}/(a_j r) \rceil W_i/L^{\frac{1}{2}})$. The first term is the cost of going around the at most α/a_j obstacles which are hit within τ_1 of the nearest corner ($\tau_1 = rW_i/L$). The second is the cost of doing at most $\lceil L^{\frac{1}{2}}/(a_j r) \rceil$ full sweeps in a corridor of height $h = W_i/L^{\frac{1}{2}}$. This gives a cost of $O(\frac{rW_i}{L} + \frac{W_i}{r\alpha})$ per unit of advance along the positive x direction because there are α/a_j units of horizontal motion while traversing a field which is not marked inactive. Thus we have a total of this type of cost in the stage of $O(\frac{nrW_i}{L} + \frac{nW_i}{r\alpha})$, using the trivial

bound of n on the horizontal distance covered in a stage. Using the values $L = n^{4/9}$, $\alpha = n^{2/3}$, and $r = n^{-1/9}$, this quantity is $O(W_i n^{4/9} + W_i n^{4/9})$, which is $o(W_i n^{4/9} \log n)$.

3. We now consider the cost incurred while performing little-τ sweeps in fields when we *do* mark the corridor inactive. The reason the analysis above fails here is that in this case we may not reach the end of the field and so we cannot guarantee an advance of α/a_j in the x-direction. Instead, we use the fact that any corridor will be marked inactive at most once in a phase.

   The number of corridors marked inactive in subphase j is at most $a_j - a_{j+1}$. Therefore in subphase j the cost of this portion is $O((a_j - a_{j+1})[(\alpha/a_j)\frac{rW_i}{L} + [L^{\frac{1}{2}}/(a_j r)]W_i/L^{\frac{1}{2}}])$. If the phase ends after S subphases, the portion of this cost in the phase is

$$O\left(\sum_{j=1}^{S}(a_j - a_{j+1})[(\alpha/a_j)\frac{rW_i}{L} + \frac{W_i}{(a_j r)}]\right),$$

   which is

$$O\left(W_i \cdot (\alpha r/L + 1/r) \cdot \sum_{j=1}^{S}((a_j - a_{j+1})/a_j)\right).$$

   By Claim 3.1 this is $O((\log L)W_i \cdot (\alpha r/L + 1/r))$ since $a_1 \le 2\lceil L^{\frac{1}{2}}\rceil + 1$. Because there are at most $32L^{\frac{1}{2}}$ phases in a stage, the total of this type of cost in the stage is $O(L^{\frac{1}{2}} W_i \log L \cdot (\alpha r/L + 1/r))$, which is $O(W_i n^{1/3} \log n)$ and hence $o(W_i n^{4/9} \log n)$.

4. The cost incurred in movement during the big-τ sweeps. This is the motion in the corridor of the chosen field, but to its left, and that of the big-τ sweeps of fill-up.

   - For each of the at most $32L^{\frac{1}{2}}$ phases and for each of the at most $2\lceil L^{\frac{1}{2}}\rceil + 1$ corridors, the robot makes at most $L^{\frac{1}{2}}$ big-τ sweeps before a corridor is declared useless. Each sweep costs $2h = 2W_i/L^{\frac{1}{2}}$ for motion that does not "go around" obstacles (i.e., when obstacles are not hit within τ_2 of the nearest corner). Thus, for each phase, this cost is $O(W_i L^{\frac{1}{2}})$. For the whole stage it is $O(W_i \cdot L)$.
   - When colliding with an obstacle within τ_2 of its nearest corner, we "go around" it, and return to a point of the same y-coordinate as that of the impact point. We must total the cost incurred going around obstacles. The regions in which we run the τ_2-sweeps in a subphase are below or above fields of corridors that we mark as non-active in the same subphase. (This is true both for the τ_2-sweeps to the left of the desired field and for the fill-up procedure.) Thus, the number of such regions is at most $a_j - a_{j+1}$, the number of corridors that are declared non-active in the subphase. The width of a field is α/a_j. Thus in subphase j of a certain phase, for this portion of the work, we pay $O((a_j - a_{j+1}) \cdot (\alpha/a_j) \cdot W_i/L)$, since $\tau_2 = W_i/L$. In a phase this sums to $O(W_i \cdot (\alpha/L) \log L)$ by Claim 3.1, and this is $O(W_i \cdot (\alpha L^{-\frac{1}{2}} \log L))$, because of the at most $32L^{\frac{1}{2}}$ phases in a stage.

   Plugging in our values of α and L yields a total cost for big-τ sweeps of $O(W_i n^{4/9} \log n)$.

Adding together the four costs yields a total cost of $O(W_i n^{4/9} \log n)$. □

We now prove the second property of the algorithm in a stage, namely:

LEMMA 3.3. *Let* OPT *denote the optimal cost, and let i be such that $W_i >$* OPT*. Then, for large enough n, with probability at least $3/4$ the robot reaches the wall by the end of stage i.*

To prove the above lemma we need several definitions and Lemma 3.4. First, fix an optimal Manhattan source-target path Q. It is not hard to see that without loss of generality we may assume that the path never goes in the negative x direction. We call a path with that property *non-x-decreasing.*

Definition. For an interval I, open, closed, or half-open, let the *weight of I* be the sum of the lengths of all *vertical* segments in Q with x-coordinate in I.

Definition. In some (random) execution of the algorithm, suppose a phase starts with the robot at point (x_s, y_s). When the (random) phase ends, let (x_f, y_f) denote the position of the robot. We say the phase is *good* if the weight of $(x_s, x_f]$ is at least $(1/2)(W_i/L^{\frac{1}{2}})$, or if $x_f = n$ (we hit the wall).

LEMMA 3.4. *Let i be such that $W_i >$* OPT*, and consider any phase of this stage. Whenever and wherever the phase starts, the probability that it will be good is at least* $1/4$.

Proof. Let (x_s, y_s) denote the position of the robot when the phase starts, and (x_f, y_f) its position when the phase ends.

Define $b_0, b_1, b_2, ...$ by $b_j = x_s + (j-1)\alpha$. The jth subphase, if it exists, starts on the line $x = b_{j-1}$ and ends on the line $x = b_j$ (unless it hits the wall or the phase ends). Let w_j be the weight of (b_{j-1}, b_j).

Choose a least $x'_f \geq x_s$ such that the weight of $(x_s, x'_f]$ is at least $(1/2)(W_i/L^{\frac{1}{2}})$, taking $x'_f = n$ if no such x'_f exists. Let us say that a *failure* occurs if $x_f < x'_f$; we will prove that the probability of a failure is at most $3/4$. This implies that the probability that the phase is good is at least $1/4$.

Consider the section P of Q starting on the line $x = x_s$ and ending on the line $x = x'_f$. If a failure occurs, then either (a) P leaves the corridor C in which it starts, or (b) P does not leave C between $x = x_s$ and $x = x'_f$, yet $x_f < x'_f$ anyway. Let us denote the probability that P leaves C by s. (The randomness here is in the random offset which defines C.)

Since x'_f is minimal, the weight of the open interval (x_s, x'_f) is at most $(1/2)(W_i/L^{\frac{1}{2}})$. Since at the beginning of the phase the robot chooses at random the offset of the corridor-boundaries in the range $[0, W_i/L^{\frac{1}{2}}]$ (i.e., at least twice the weight), the probability that P crosses a boundary is at most $1/2$. Thus $s \leq 1/2$.

Now we must bound the probability that (b) P does not leave C between $x = x_s$ and $x = x'_f$, yet $x_f < x'_f$. Since P does not leave C between $x = x_s$ and $x = x'_f$, there cannot be any obstacle extending above and below C whose left edge is on the line $x = x_s$, or to the right of this line and strictly to the left of the line $x = x'_f$.

Suppose the phase ends with $x_f < x'_f$. Because the phase ended, all corridors must have been made non-active. This includes corridor C. Corridor C cannot be declared useless since by Claim 2.1, if C is declared useless then any path, such as P, which starts at $x = x_s$ and ends strictly to the right of the line $x = x_f$ and stays within C, must have vertical cost at least $L^{\frac{1}{2}} \cdot \tau_2 = L^{\frac{1}{2}} \cdot W_i/L = W_i/L^{\frac{1}{2}}$ in the interval (x_s, x_f), and this is more than the cost of P in the interval (x_s, x'_f). Thus we can conclude that, when the phase ended, C is *not* marked useless. This implies both that the phase ended when the robot reached the end of a width-α strip of some subphase (as opposed to in the middle of such a strip), and that corridor C has been declared inactive.

We consider the probability that C becomes inactive during some subphase j such that $b_j \leq x'_f$ (such a subphase begins at $x = b_{j-1}$ and ends at $x = b_j$). Define w_j to be the weight of (b_{j-1}, b_j). We argue that the probability that C becomes inactive in subphase j is at most $w_j L^{\frac{1}{2}}/W_i$. This follows since if we make C inactive then we have made at least $\frac{1}{r}(L^{\frac{1}{2}}/a_j)$ full small-τ sweeps in the chosen field of C. Claim 2.1 implies that the weight of the open interval defined by the vertical boundaries of the field is at least $[\frac{1}{r}L^{\frac{1}{2}}/a_j] \cdot [rW_i/L] = W_i/(L^{\frac{1}{2}}a_j)$. Within subphase j, there can be at most $w_j L^{\frac{1}{2}} a_j/W_i$ such fields of weight at least $W_i/(L^{\frac{1}{2}}a_j)$ in the corridor C. In C, the uniform random choice performed at the beginning of a subphase is also a uniform random choice of the field to visit in C. Therefore, the probability that we mark the corridor inactive within the subphase is the chance that one of those at most $w_j L^{\frac{1}{2}} a_j/W_i$ fields out of the a_j fields is chosen. This probability is at most $w_j L^{\frac{1}{2}}/W_i$.

Conditioning on the fact that P lies entirely in C, let R be the event that C is marked inactive before x'_f and R_j the event that C is marked inactive in subphase j, for j such that $b_j \leq x'_f$. Then $R = \cup R_j$ and $P[R_j] \leq w_j L^{\frac{1}{2}}/W_i$, where w_j is the weight of (b_{j-1}, b_j). Thus,

$$\begin{aligned} P[R] &\leq \sum_j P[R_j] \\ &\leq \sum_j w_j L^{\frac{1}{2}}/W_i \\ &= (L^{\frac{1}{2}}/W_i)\sum_j w_j \\ &< (L^{\frac{1}{2}}/W_i)[(1/2)(W_i/L^{\frac{1}{2}})] \\ &= \frac{1}{2}\,. \end{aligned}$$

Here we used $\sum_j w_j < \frac{1}{2}W_i/L^{\frac{1}{2}}$, because the weight of (x_s, x'_f) is assumed to be less than $\frac{1}{2}W_i/L^{\frac{1}{2}}$.

To conclude, the probability that P does not leave C is $1 - s$. If P does not leave C, then the conditional probability that $x_f < x'_f$ is at most $1/2$. It follows that the probability that P doesn't leave C and $x_f < x'_f$ is at most $(1-s)(1/2)$. Therefore, the probability of failure is at most s (the probability that P leaves C), plus $(1-s)(1/2)$ (an upper bound on the probability that P stays in C and $x_f < x'_f$). But $s \leq 1/2$ implies that $s + (1-s)(1/2) \leq 3/4$. □

We can now prove Lemma 3.3. The only way a stage can fail to reach the wall is that there are fewer than $2L^{\frac{1}{2}}$ good phases of the $32L^{\frac{1}{2}}$ in the stage, since if there were at least $2L^{\frac{1}{2}}$ good phases in the stage, the optimal cost in those $2L^{\frac{1}{2}}$ good phases would be at least $2L^{\frac{1}{2}}[(1/2)W_i/L^{\frac{1}{2}}] = W_i > \text{OPT}$. Considering the phases as a series of Bernoulli trails, we have $m = 32L^{\frac{1}{2}}$ trials, each of which has, regardless of what happened in the past, probability at least $1/4$ of being good. We want to upper-bound the probability of having fewer than $2L^{\frac{1}{2}} = am = (1/16)m$ successes. Different phases are not independent, but since each is good with probability at least 1/4 regardless of the past, our chance of failure is at most what it would be if they were independent and each had probability exactly 1/4 of being good (e.g., see [GS], p.490). We use the following Chernoff bound (cf. [HR]) to bound the probability of this event

$$Pr[S \leq am] \leq [(\frac{p}{a})^a e^{a-p}]^m \ .$$

As $p \geq 1/4$, we have

$$\begin{aligned} Pr[S \leq am] &\leq [(\frac{p}{a})^a e^{a-p}]^m \\ &\leq [(\frac{1}{a})^a e^{a-\frac{1}{4}}]^m \\ &= [16^{1/16} \cdot e^{-3/16}]^m \\ &= [16^{1/16} \cdot e^{-3/16}]^{32L^{\frac{1}{2}}}. \end{aligned}$$

Since $16^{1/16} \cdot e^{-3/16} < 1$, this is less than 1/4 for large enough $L^{\frac{1}{2}}$, i.e., large enough n. □

4 Point-to-Point Navigation

We can use our randomized algorithm for the wall problem to obtain a randomized algorithm with the same competitive ratio for point-to-point navigation, using a technique from [BRS]. Thus, our point-to-point algorithm is also provably better than any deterministic algorithm. For the sake of completeness, we sketch below the [BRS] technique. Let us say that the robot has to reach the point $t = (n_x, n_y)$ from $s = (0, 0)$, and let $n = \max\{n_x, n_y\}$.

The robot takes a greedy "move rightwards if possible, else up" path towards t until it reaches some point $s' = (x_{s'}, y_{s'})$ such that either $x_{s'} = n_x$ or $y_{s'} = n_y$. Assume without loss of generality that $y_{s'} = n_y$. The robot now uses the wall-problem algorithm to reach a point on the infinite line $x = n_x$; say it reaches (n_x, y_0). Without loss of generality assume that $y_0 \geq n_y$. The robot now takes a greedy "move down if possible, else left" path from (n_x, y_0) until its y-coordinate equals n_y. The fact that our wall-problem algorithm is non-x-decreasing guarantees that this new greedy path reaches a point (x_0, n_y) with $x_0 \geq x_{s'}$. The greedy (monotone) path connecting (x_0, n_y) and (n_x, y_0) is now used to run the room-problem algorithm of [BRS, BBFY] (cf. [BRS], Theorem 5). We can now prove:

THEOREM 4.1. *There is a randomized point-to-point navigation algorithm with competitive ratio of $O(n^{\frac{4}{9}} \log n)$.*

Proof Sketch. Denote by OPT the length of the optimal path from s to t. Clearly OPT $\geq n$. Also, there exists a path from s' to the "wall" $x = n_x$ of length at most $n +$ OPT. Therefore, invoking our randomized algorithm for the wall problem, the robot goes along a path of *expected* length $O(\text{OPT} \cdot n^{\frac{4}{9}} \log n)$. Furthermore, by the same arguments as in the proof of Theorem 3.1, the expected distance $|y_0 - n_y|$ is $O(\text{OPT})$. Therefore the expected distance traveled by the robot when applying the deterministic room-problem algorithm is $O(\text{OPT} \cdot \log n)$ (cf. [BRS] Theorem 5, [BBFY]). In addition the robot took several greedy paths of total length $O(n)$. Thus the total expected distance the robot travels is $O(\text{OPT} \cdot n^{\frac{4}{9}} \log n)$. □

5 Navigating in Three Dimensions

Using the same general techniques as for the 2-dimensional case, we also have an $O(n^{2/3-\epsilon})$-competitive randomized algorithm for robot navigation in three dimensions (omitted due to space considerations). The lower bound for deterministic robot navigation algorithms in three dimensions is $\Omega(n^{2/3})$.

References

[BCR] R.A. Baeza-Yates, J.C. Culberson, and G.J.E. Rawlins. Searching in the Plane. *Information and Computation*, 1993.

[BBFY] E. Bar-Eli, P. Berman, A. Fiat, and P. Yan. On-line Navigation in a Room. In *Proc. 3rd ACM-SIAM SODA*, 1992.

[BBKTW] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson. On the Power of Randomization in Online Algorithms. In *Proc. of the 22nd Ann. ACM Symp. on Theory of Computing*, pp. 379–386, May 1990.

[BC] A. Blum, P. Chalasani. An On-line Algorithm for Improving Performance in Navigation. In *Proc. 34th IEEE Annual Symp. on Foundations of Comp. Science*, pp. 2–11, 1993.

[BRS] A. Blum, P. Raghavan, and B. Schieber. Navigating in Unfamiliar Geometric Terrain. In *Proc. 23rd ACM STOC*, 1991.

[BLS] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. In *Proc. 19th Annual ACM Symposium on Theory of Computing*, pp. 373-382, 1987.

[Ch] P. Chalasani. Online Performance-Improvement Algorithms. Ph. D thesis. CMU tech report CMU-CS-94-179. August 1994.

[DKP] X. Deng, T. Kameda, and C. Papadimitriou. How to Learn an Unknown Environment I: The Rectilinear Case. Tech. Report CS-93-04, Dept. of Comp. Science, York University.

[FKLMSY] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator, and N.E. Young. Competitive Paging Algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[GS] Y. Gurevich and S. Shelah. Expected computation time for Hamiltonian path problem. *SIAM J. Comput*, 16(3) pp. 486–502, 1987.

[HR] T. Hagerup, and C. Rüb. A Guided Tour of Chernoff Bounds. *Information Processing Letters* 33 (1990), pp. 305–308.

[Kl] J. Kleinberg. On-Line Search in a Polygon. In *Proc. 5th ACM-SIAM SODA*, pp. 8–15, 1994.

[KRR] H. Karloff, Y. Rabani, and Y. Ravid. Lower Bounds for Randomized k-Server and Motion-Planning Algorithms. *SIAM Journal on Computing* 23 (1994), 293-312.

[KRT] M. Kao, J. Reif, and S. Tate. Searching in an Unknown Environment: An Optimal Randomized Algorithm for the Cow-Path Problem. In *Proc. 4th ACM-SIAM SODA*, pp 304–313, 1993.

[La] J. -C. Latombe. *Robot Motion Planning*, Kluwer Academic Publishers, Boston, 1991.

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

[PY] C. Papadimitriou, and M. Yannakakis. Shortest Paths Without a Map. *Theoretical Computer Science*, 84(1991), pp. 127–150.

[Sh] M. Sharir. Algorithmic motion planning in robotics, *Computer* **22**, March 1989, pp. 9–20.

[ST] D.D. Sleator and R.E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communication of the ACM*, 28(2) pp. 202–208, 1985.

6 Appendix: A lower bound for "x-oblivious" strategies

Let us say that an algorithm is *x-oblivious* if whenever it is possible for the robot to move in the positive x direction, the robot does so and *does not keep track of the x-distance moved.* In other words, the memory of the robot may be thought of as a list of events of the form:

> "First I hit an obstacle extending from y_1 to y'_1, then I moved up around it, then I moved forward, then I hit an obstacle extending from y_2 to y'_2, then I moved down around it, then I moved forward, etc."

For instance, the deterministic algorithm presented in [BRS] can be viewed as acting in this manner. We show here that any x-oblivious randomized strategy cannot beat the deterministic lower bound.

THEOREM 6.1. *Any x-oblivious randomized algorithm for the wall problem has competitive ratio $\Omega(\sqrt{n})$.*

Thus, to take advantage of randomization, one must explicitly keep track of one's x-coordinate (which our algorithm does).

Proof. Consider a "complete brick pattern" of obstacles each having width 1 and height $h = 2\sqrt{n}$. Specifically, in a complete brick pattern there are obstacles with lower left corner at each point of the form $(i, (i/2+j)h)$, for all integers i, j, with $0 \leq i \leq n$. Now, pick some random $j \in \{-2\sqrt{n}, \ldots, -1, 0, 1, \ldots, 2\sqrt{n}\}$ and remove all but every $\sqrt{n}$-th obstacle whose center lies on the line $y = j\sqrt{n}$. In other words, out of the $n/2$ obstacles originally in existence with center at $y = j\sqrt{n}$, only $\sqrt{n}/2$ are left and they are equally spaced. Notice that in this scene there exists an $O(n)$-length path from the start to the wall. We prove this defines a hard case for the algorithm.

Since the adversary's (randomized) strategy is fixed, by standard min-max arguments we may assume that the algorithm is deterministic. Let P be the path taken by the algorithm on the *original* complete brick pattern (before obstacles were removed). Since the algorithm always moves rightward when possible, the path P hits exactly n obstacles in the complete brick pattern, and in fact we may think of P as a sequence of n letters $(u, u, d, d, d, u, d, \ldots)$ indicating whether the algorithm moves *u*p or *d*own at each obstacle. Consider now the new obstacle scene (the one in which some obstacles have been removed). Because the algorithm is x-oblivious, its sequence of up/down motions is *exactly the same* as for the complete brick pattern, only it may finish earlier depending on how often it reaches y-coordinate $j\sqrt{n}$. Because j was picked randomly from $4\sqrt{n}$ choices, we expect the algorithm to visit that y-coordinate only $\sqrt{n}/4$ times in path P. Thus, in the new pattern, the expected "free" x-motion by the algorithm is at most $(\sqrt{n}/4)(2\sqrt{n}) = n/2$, and the algorithm's expected cost remains $\Theta(n^{3/2})$, which is $\Theta(\sqrt{n})$ times the optimal cost. □