On-Line Algorithms for Robot Navigation and Server Problems

by

Jon Michael Kleinberg

A.B., Computer Science and Mathematics
Cornell University
(1993)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Signature of Author______________________________________
Department of Electrical Engineering and Computer Science
May 5, 1994

Certified by______________________________________
Michel X. Goemans
Assistant Professor of Applied Mathematics
Thesis Supervisor

Accepted by______________________________________
F.R. Morgenthaler
Chairman, Department Committee on Graduate Students

On-Line Algorithms for Robot Navigation and Server Problems

by

Jon Michael Kleinberg

Submitted to the Department of Electrical Engineering and Computer Science

on May 5, 1994, in partial fulfillment of the

requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

Abstract

Many classical problems of computer science — such as paging, scheduling, and maintaining dynamic data structures — are naturally *on-line*; an algorithm for such a problem is constantly making irrevocable decisions without knowing what its future input will be. The *competitive analysis* of on-line algorithms was brought into prominence by the work of Sleator and Tarjan in 1985 as a theoretical framework in which to measure the performance of such algorithms. Since then, a variety of on-line problems have been studied from this perspective.

We consider two such problems in this setting: robot navigation, and computationally efficient algorithms for the k-server problem of Manasse, McGeoch, and Sleator. For the first of these, we give algorithms for a robot searching for a goal in an unknown simple polygon; our approach can be viewed as an extension of some of the strategies of Baeza-Yates, Culberson, and Rawlins to a more general class of geometric search spaces. We also provide the first competitive analysis of the *robot localization problem* — a fundamental task for an autonomous mobile robot, in which it must determine its location in a known environment.

Finally, we address the general question of how the performance of an on-line algorithm varies with the computational resources it has at its disposal. Within the context of the k-server problem, a natural specialization of this question is the following: can an algorithm which uses only constant space, and constant time in processing each request, match the performance of algorithms which are allowed to perform an arbitrary amount of computation at each step? We show a non-trivial lower bound for the class of 2-server balancing algorithms, a subclass of the constant-time algorithms; this represents one of the first lower bounds for an on-line algorithm based solely on its computational resources.

Thesis Supervisor: Michel X. Goemans

Title: Assistant Professor of Applied Mathematics

Keywords: On-line algorithms, robot navigation, robot localization, k-server problem

Acknowledgements

First of all, I would like to thank Éva Tardos, with whom my work in this area began, for numerous helpful ideas and conversations. The thesis benefitted enormously from her enthusiasm for and interest in thinking about the problems I was working on. At MIT, Michel Goemans supervised this thesis; I thank him for many interesting discussions on this subject, for his consistently valuable suggestions, and for his careful reading of this material as it assumed its final form.

The problems considered in Chapter 4 grew out of discussions with Dan Huttenlocher at Xerox PARC; I would like to thank Dan for his unfailing interest in whatever I was working on, and for frequently helping me to identify the truly interesting problems in this and related areas. In this regard, thanks also to Bruce Donald and all the regular attendees of the "Robot Cafe" seminar at Cornell, for many helpful conversations; to Leo Guibas, who first suggested the robot localization problem to me; to Ronitt Rubinfeld, Michal Parnas, and Dana Ron for sharing with me their ideas on robot navigation; and to Tony Yan, who was endlessly patient in listening to my thoughts on these problems, and always seemed to ask the right questions. Allan Borodin's class in the fall of 1993 provided a very interesting perspective on on-line algorithms; the presentation in Section 2.1 and Chapter 5 reflects this influence.

Section 5.1 is joint work with Ran El-Yaniv, who visited MIT in the fall of 1993. I am also grateful to Rolf Klein, who pointed out an error in an earlier version of the algorithm in Section 3.1.

This work has been partially supported by the Sloan Fellowship and NSF PYI Award of Éva Tardos, and subsequently by an ONR Graduate Fellowship.

Contents

1 Introduction **9**

2 On-Line Algorithms: An Overview **13**

2.1 Competitive Analysis 14

2.2 The k-Server Problem 16

2.3 Other On-Line Problems 20

2.4 Robot Navigation 21

3 Searching an Unknown Polygon **26**

3.1 Traversing an Unknown Street 27

3.2 Exploring a Rectilinear Polygon 34

3.3 Searching a Rectilinear Polygon 38

4 The Robot Localization Problem **42**

4.1 Lower Bounds and Other Examples 47

4.2 The Algorithm for Trees 49

4.3 The Algorithm for Rectangles 57

4.4 Placing Unique Landmarks 61

5 Real-Time Server Algorithms **65**

5.1 Two Servers in Euclidean Space 67

5.2 A Lower Bound for Balancing Algorithms 71

6 Conclusion and Open Problems **77**

6.1 Robot Navigation 77

6.2 Servers 81

Chapter 1

Introduction

The distinction between on-line and off-line algorithms is an old one in computer science, yet the current surge of on-line algorithms research in the theory community is relatively new. In large part this is due to the surprisingly rich structure of the *competitive ratio*, brought into focus by Sleator and Tarjan [ST] in 1985 as a means of analyzing on-line algorithms.

At a general level, an on-line algorithm is one which receives its input in a piecemeal fashion; at each step, it must make irrevocable decisions before seeing the remainder of the input. An off-line algorithm, on the other hand, has the luxury of reading the entire input before performing any computation. The competitive ratio of an on-line algorithm is then defined to be the worst-case ratio of the cost it incurs on a given problem instance to the cost of the optimum (off-line) solution. (Readers not familiar with the definitions in this area should read Chapter 2 before this Introduction.)

In this thesis, we consider two naturally on-line problems within the framework of competitive analysis: robot navigation, and the k-server problem.

In a typical on-line navigation problem, a robot with vision is placed in a geometric environment for which it does not have a map; it must perform some task in this environment, such as constructing a map, or finding a recognizable goal hidden in the environment. Baeza-Yates, Culberson, and Rawlins [BCR], in one of the first papers in this area, considered the latter type of *search problem.* They gave algorithms for searching in a number of structured situations, including the problem of searching for a point on a line, a line in the plane, and a point in a grid graph. The basis for these algorithms is a collection of techniques that have found wide application in subsequent papers.

In Chapter 3, we consider more geometric versions of this search problem — specifically the problem of a robot searching for a goal in an unknown rectilinear polygon. This differs from the model of [BCR] in that the structure of the space to be searched is not known to the robot; the generality of rectilinear polygons also makes possible a much wider class of search spaces. The search algorithm we give is fairly natural, performing exploration on larger and larger regions of the polygon. This problem of exploration in a simple rectilinear polygon has been considered separately by Deng, Kameda, and Papadimitriou [DKP], who gave an algorithm with a competitive ratio of 2; we offer a simple randomized variant with competitive ratio 5/4.

It is also interesting to consider special cases of the search problem in a polygon — restricting the type of polygon so that better competitive ratios can be obtained. This was the approach taken by Klein [K], who considered the class of *streets* and gave a search algorithm with competitive ratio at most $1 + \frac{3}{2}\pi$ (~ 5.71). We give an algorithm for streets with competitive ratio at most $\sqrt{4 + \sqrt{8}}$ (~ 2.61), improving on this bound by more than a factor of two. A number of other types of polygons may well be amenable to the same type of approach, and this appears to be an interesting direction for further work.

Not all on-line navigation problems assume a complete lack of information about the environment, however. For example, a fundamental task for an autonomous mobile robot is that of *localization* — determining its location in a *known* environment. This problem arises in settings that range from the computer analysis of aerial photographs to the design of autonomous Mars rovers. Typically, localization occurs in two phases: first the robot determines all possible locations consistent with its current local view of the environment; then it moves around, performing enough "reconnaissance" to determine uniquely where it is. Despite the attention that localization has received in the robotics literature, the first theoretical work on it appeared only recently in a paper of Guibas, Motwani, and Raghavan [GMR]; they gave geometric algorithms for the first part of the problem — enumerating locations consistent with a view — but did not address the issue of strategies for the second part.

In Chapter 4, we consider this aspect of localization, providing algorithms for a mobile robot to move around in its environment so as to determine efficiently where it is. We argue that localization is a natural setting in which to apply the competitive analysis of on-line algorithms;

the distance traveled by the robot is compared to the length of the shortest possible localizing tour. The search strategies of [BCR] allow one to obtain a fairly straightforward, and non-optimal, localization algorithm; our main result here is an algorithm which makes stronger use of the robot's knowledge of the map in order to improve asymptotically on this approach. An interesting feature of our technique is the way in which the robot is able to identify "critical directions" in the environment which allow it to perform late stages of the search more efficiently.

Finally, in Chapter 5, we consider some problems related to the computational resources required by an on-line algorithm. The overwhelming majority of research in competitive analysis has proceeded along information-theoretic lines; that is, the only factor limiting the performance of an on-line algorithm is its lack of information about the nature of future input. But for any given problem, a natural question to ask is whether the best competitive ratio attainable by an on-line algorithm depends on the computational resources it has at its disposal. We consider such questions within the context of the k-server problem — an elegant generalization of paging and certain types of scheduling problems, and one of the most well-studied problems in the area of on-line algorithms. Here, an algorithm is presented with a metric space M and k mobile *servers* that can move within M. Points of M request service, one after another, and the algorithm must move a server to the location of each request before seeing the next one. The goal is to minimize the total distance traveled by the servers. (See Chapter 2 for background on the k-server problem.)

We say that a *real-time* server algorithm is one which uses only a constant amount of space, and constant time per request. This notion has been studied especially with respect to the 2-server problem, by Irani and Rubinfeld [IR] and Chrobak and Larmore [CL2]. In this setting it has been observed that, while there are computationally expensive 2-server algorithms achieving the optimal competitive ratio of 2 [MMS, CL1, CL4], the best known competitive ratio for a real-time algorithm is 4 [CL2].

In Chapter 5, we give a real-time 2-server algorithm which is 2-competitive in n-dimensional space under the L_1 ("Manhattan") metric. This considerably extends the class of metric spaces known to have optimal real-time algorithms; the technique we use can be seen as a higher-dimensional generalization of the elegant "Double-Coverage" algorithm discovered for k servers on a line by Chrobak, Karloff, Payne, and Vishwanathan [CKPV]. We also prove a lower bound for 2-server balancing algorithms, a sub-class of the real-time algorithms. Balancing algorithms have been proposed for a number of special cases of the k-server problem; they are so named because they seek to "balance" the distance traveled evenly among the servers. We show that no such algorithm can achieve a competitive ratio better than $(5+\sqrt{7})/2$ (~ 3.82) for the 2-server problem; this shows that no optimal on-line 2-server algorithm can be expressed as a balancing algorithm. This also represents, to our knowledge, one of the first lower bounds for an on-line algorithm based solely on its computational resources.

Some Notation

We use standard order-of-growth notation (O, Ω, Θ) throughout; additionally we say that $f(n)$ is $o(g(n))$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

and $f(n)$ is $\omega(g(n))$ if g is $o(f)$.

Two different metrics in $\mathbf{R}^d$ will be used at various places: L_1 and L_2. In the L_1 ("Manhattan") metric, the distance between points $x = (x^1, \ldots, x^d)$ and $y = (y^1, \ldots, y^d)$ is given by

$$L_1(x, y) = \sum_{i=1}^{d} |x^i - y^i|.$$

The L_2 metric is the standard Euclidean distance; that is,

$$L_2(x, y) = \left(\sum_{i=1}^{d} (x^i - y^i)^2 \right)^{1/2}.$$

Chapter 2

On-Line Algorithms: An Overview

The study of on-line algorithms stems from the issue of how an algorithm receives its input. In the traditional view, all input is present immediately: for example, if the algorithm is supposed to be computing the shortest path between two nodes in a graph, then one assumes that it has access to entire graph from the start. This is clearly a reasonable assumption in a large number of situations; however, there are equally natural situations in which it makes sense to imagine that the algorithm is receiving its input in an *on-line* fashion: the input arrives in a sequence of small pieces, and with the arrival of each piece, the algorithm must make certain immediate and irrevocable decisions.

In this sense, there is nothing new about on-line algorithms. Certainly, the difference between "one-pass" and "two-pass" algorithms has always been a fundamental issue in designing computer programs; and this is essentially the same distinction expressed in different terminology. In other cases, the circumstances of the problem force us to work within the on-line framework — for example, paging algorithms evict pages from memory, call-control algorithms route connections through a network, and trading strategies tell investors to buy and sell, all of them without knowing what the future sequence of "input" will look like. For such problems, there are only on-line algorithms.

At the same time, however, there has been a tremendous increase in the number of "on-line algorithms" papers over the last ten years; this has derived mainly from the increasing popularity of *competitive analysis* as a perspective from which to discuss the performance of such algorithms. In the following section, we discuss competitive analysis in general, and specifically the paper of Sleator and Tarjan [ST].

2.1 Competitive Analysis

Typically, an on-line algorithm is trying to solve some optimization problem (as in the examples cited above). Thus, if $\mathcal{A}$ is an on-line algorithm, and σ is an input sequence, then the crucial parameter which must be minimized in most on-line problems is $\mathcal{A}(\sigma)$, the *cost* incurred by $\mathcal{A}$ on σ.

For any given problem, we can consider the optimal *off-line* algorithm OPT, which sees the entire input ahead of time and thus always achieves the minimum possible cost on σ. In the traditional analysis of algorithms, the fundamental question is how computationally "expensive" (in terms of time or space complexity) it is to design the algorithm OPT. In the competitive analysis of on-line algorithms, one instead takes OPT as a given and asks how close an on-line algorithm $\mathcal{A}$ can come to achieving the performance of OPT.

Thus we say that an on-line algorithm $\mathcal{A}$ is c-competitive if for some absolute constant β, the expression

$$\mathcal{A}(\sigma) - c \cdot OPT(\sigma)$$

is bounded by β for all input sequences σ. We will sometimes say that $\mathcal{A}$ is "competitive" if it is c-competitive for some constant c; the infimum of the set of c for which $\mathcal{A}$ is c-competitive is its *competitive ratio*.

From a historical point of view, it is quite difficult to identify the first use of this performance measure as applied to on-line algorithms. It appears in the bin-packing literature of the 1970's, which considered the well-known problem of determining the fewest number of "bins," each of capacity 1, which are required to contain a set of n objects with weights $w_1, \ldots, w_n \leq 1$. This problem is NP-Complete, but a number of very natural heuristics, several of them on-line algorithms, give constant-factor approximations to the minimum number of bins required.

In this setting, an on-line algorithm is simply one which decides where to place the i^{th} object before seeing the values of $w_{i+1}, \ldots, w_n$. One such example is *First Fit*, which puts the object in the first bin in which it will fit, starting a new bin only if necessary. Garey, Graham, and Ullman [GGU] proved that the competitive ratio of First Fit is $\frac{17}{10}$, and an almost exact characterization of the performance of First Fit, as well as a number of other approximation algorithms, was given by Johnson in his Ph.D. thesis [J]. In a result very much in the spirit of current work in on-line algorithms, Yao proved in 1980 that there is no on-line bin-packing algorithm with a performance ratio better than $\frac{3}{2}$ [Yao].

The competitive ratio also appears sporadically (and again not cast in the current terminology) in work on graph-coloring. The First Fit algorithm has a natural meaning in this world as well: proceed through the vertices one at a time, coloring each with the lowest-numbered color possible. While First Fit performs abysmally on some classes of graphs, it has long been observed to have relatively good behavior on others. The survey paper of Kierstead and Trotter [KT] mentions a number of problems related to the First Fit coloring algorithm; for example, Woodall asked in 1974 whether the number of colors used by First Fit was within a constant factor of $\chi(G)$ when G is an interval graph (see also [Kier]). In the new language, this is simply the question of whether First Fit achieves a constant competitive ratio for this type of graph. In the last five years, there have been a number of papers on "on-line graph coloring" in the competitive framework; these will be discussed briefly in Section 2.3.

But it was the work of Sleator and Tarjan in 1985 [ST] which launched the current deluge of papers. [ST] considered two classical problems of computer science, within the setting of competitive analysis: the maintenance of a dynamic data structure, and paging.

On the first of these questions, [ST] dealt specifically with the problem of maintaining a linked list under a sequence of *insert*, *delete*, and *member?* requests. This problem had a long history, and many heuristics had been proposed. In the context of on-line algorithms, as in the actual applications, one is trying to minimize the total number of memory references over the course of a request sequence (i.e. one always wants the current item referenced to be as close to the front of the list as possible). What Sleator and Tarjan showed was that the well-known heuristic MOVE-TO-FRONT (always put the item just accessed at the front of the list) is 2-competitive. One thing that makes this result particularly striking is that the competitive ratio of such a simple rule should be so low, especially when there is in fact no good characterization known for the optimal off-line algorithm. The proof was also interesting for its use of a *potential function* argument, a technique that was to become extremely common in subsequent research in on-line algorithms (we will give an example of such an argument in Section 2.2).

The results in [ST] concerned with paging gave much stronger lower bounds. Here one pictures a computer with k pages of fast memory and an unlimited number of additional pages of slow memory (e.g. a disk). When a reference is made to a page not in fast memory, a "page fault" occurs: the referenced page must be moved in, and some page currently sitting in fast memory must be swapped out. The goal is to minimize the number of page faults, over the course of a string σ of memory references. In [ST] it was proved that no on-line paging algorithm with k pages can be better than k-competitive, and a number of well-known algorithms such as LRU (evict the least-recently used page) and FIFO (evict the page that was brought in longest ago) match this bound. Of course, from the point of view of operating system design, a performance guarantee of k is ridiculously large — a rule such as LRU actually performs extremely well in practice — but the result does lend some insight into why looking at previous requests should help in designing a paging algorithm. Various extensions to the model of [ST] have been proposed [BIRS, KPR], but the question of why on-line paging works so well in real life remains an intriguing one.

Following [ST], the paper [KMRS] analyzed additional on-line strategies for cache management (which is essentially the same as paging in this model), and [BLS] proposed "metrical task systems" as an abstract model for studying on-line algorithms. The following year, Manasse, McGeoch, and Sleator [MMS] introduced what has become perhaps the most well-known and well-studied on-line problem: the k-server problem.

2.2 The k-Server Problem

We imagine the following situation. An on-line algorithm $\mathcal{A}$ controls k mobile *servers* which are free to move around in some metric space M. A finite request sequence σ is now presented to the algorithm, one request at a time. Each request is a point in the space M; the algorithm must move one of the servers to this point before seeing the next request. The goal is to minimize the total distance traveled by the servers, over the entire request sequence. The k-server problem generalizes paging in the following sense: if M is the set of possible pages of memory, with the *uniform metric* (all non-trivial distances are 1), and the servers are the k "slots" of fast memory, then the problem becomes that of minimizing the number of page faults.

In their original paper, Manasse, McGeoch, and Sleator showed that for every metric space

M and every on-line algorithm $\mathcal{A}$, there are arbitrarily long request sequences on which

$$\mathcal{A}(\sigma) \geq k \cdot OPT(\sigma) - O(1).$$

That is, *no* on-line algorithm can be better than k-competitive in *any* metric space. Note that LRU or FIFO provides a matching upper bound for the uniform metric space; [MMS] also provided a 2-competitive 2-server algorithm and a k-competitive k-server algorithm for any metric space with only $k+1$ points. Based on these special cases, they advanced the *k-server conjecture*: for every metric space M, there is a k-competitive k-server algorithm. At the time of this writing, the conjecture is still open; however, significant progress has recently been achieved in [KP].

One of the tantalizing aspects of this problem is that most natural algorithms — for example, the greedy algorithm (move the closest server) as well as most straightforward generalizations of LRU — do not achieve any bounded competitive ratio in general. Indeed, it was not known initially whether one could even achieve a constant competitive ratio (depending only on k) in an arbitrary metric space, even for the case of three servers. This was settled by Fiat, Rabani, and Ravid [FRR], who gave a general algorithm with a competitive ratio of at most $2^{O(k \log k)}$. Grove improved this bound to $2^{O(k)}$ [Gr] using a simple randomized algorithm, and derandomization techniques of Ben-David et. al. [BBKTW]. And very recently, Koutsoupias and Papadimitriou [KP] have shown that the "work-function algorithm" proposed by Chrobak and Larmore [CL4], as by well as other researchers, is at most $(2k-1)$-competitive.

In a separate direction, a number of papers extended the set of metric spaces for which good server algorithms were known. Chrobak, Karloff, Payne, and Vishwanathan [CKPV] gave an elegant k-competitive algorithm for k servers on the line (i.e. $M = \mathbf{R}^1$). The algorithm works as follows: let $[a, b]$ denote the interval on the line whose left endpoint is the leftmost server and whose right endpoint is the rightmost server. If the request falls outside this interval, move only the closest server; otherwise, move the two "neighboring" servers at the same speed toward the request until the closer one reaches it. Subsequently, Chrobak and Larmore showed that a natural generalization of this algorithm for the case in which M is a tree is also k-competitive [CL3].

The proof that this algorithm is k-competitive makes use of a short, if somewhat mysterious, potential function argument. We describe the proof for the case of two servers on a line, since it provides a good example of how such a proof proceeds.

A potential function Φ is a function defined on the set of possible configurations of the on-line and off-line servers. Thus, when any server moves, the value of Φ may change. We denote the i^{th} request in the request sequence σ by σ_i, and the cost incurred by $\mathcal{A}$ and OPT on this single request by $\mathcal{A}(\sigma_i)$ and $OPT(\sigma_i)$ respectively. Consider the following sequence of events: for each $i = 1, 2, \ldots$, first OPT serves the request σ_i, and then $\mathcal{A}$ serves σ_i. We will watch how the value of Φ changes over the course of these movements of the servers. Suppose we can show the following three facts:

1. Φ is always non-negative.

2. There is a constant c such that whenever OPT serves a request σ_i, Φ goes up by at most $c \cdot OPT(\sigma_i)$.

3. When $\mathcal{A}$ serves a request σ_i, Φ decreases by at least $\mathcal{A}(\sigma_i)$.

Note that over the course of the whole request sequence, the total of the increases in Φ is at most $c \sum_i OPT(\sigma_i) = c \cdot OPT(\sigma)$, and similarly the sum of the decreases in Φ is at least $\sum_i \mathcal{A}(\sigma_i) = \mathcal{A}(\sigma)$. Thus, if we let Φ_0 denote the value of Φ on the initial configuration of the servers, then the fact that Φ is always non-negative implies

$$\mathcal{A}(\sigma) \leq c \cdot OPT(\sigma) + \Phi_0.$$

That is, $\mathcal{A}$ is c-competitive.

It is not difficult to show (see [MMS]) that we can assume OPT is a *lazy* algorithm: it always moves at most one server on each request, and does not move if it already covers the requested point. If x and y are points on the line, let xy denote the distance between them. For the case of two servers in $M = \mathbf{R}^1$, let s_1, s_2 denote the positions of algorithm $\mathcal{A}$'s servers, o_1, o_2 the positions of OPT's servers, and M_{min} the minimum of $s_1o_1 + s_2o_2$ and $s_1o_2 + s_2o_1$ (i.e. it is the minimum-cost matching between the servers of $\mathcal{A}$ and OPT.) The potential function used in [CKPV] is an adaptation of one used by Coppersmith, Doyle, Raghavan, and Snir [CDRS];

for our purposes, we can write it as

$$\Phi = 2M_{min} + s_1 s_2.$$

Clearly Φ is always non-negative. When OPT moves a distance d, it cannot affect the second term, and it increases the first term by at most $2d$. On a request in which $\mathcal{A}$ moves only one server, say a distance d, the first term will go down by $2d$ and the second will go up by d. On a request in which $\mathcal{A}$ moves both servers a distance d', the first term cannot increase, while the second goes down by $2d'$. (Verifying each of these facts is a fairly straightforward exercise.) As this shows that Φ has the required three properties, the proof is complete.

In this thesis, we will only be considering 2-server algorithms. Note that in this setting, there is no "k-server conjecture" — it has been known since the origin of the problem that there is a 2-competitive, on-line algorithm for two servers in a general metric space, and that no on-line algorithm can achieve a better ratio. (Other 2-competitive 2-server algorithms, based on quite different techniques, have been given by Chrobak and Larmore in [CL1, CL4].) Thus the emphasis here has been to give competitive algorithms that are as "simple" as possible — simple both in the effort required to analyze them, and in the amount of computation they must perform on each request.

Specifically, all known optimal 2-server algorithms use more than a constant amount of time and space per request; indeed, when deciding which server to send to a point in M, they must perform computations based on the entire set of points constituting the previous requests. Given the nature of the problem, it might not have been unreasonable to have conjectured that this much computation is required to achieve any constant ratio. This is not the case, however. Irani and Rubinfeld [IR] gave the first competitive 2-server algorithm which performs only a constant amount of work on each request; its competitive ratio is known to be somewhere between 6 and 10. A "fast" 2-server algorithm with a competitive ratio of 4 was subsequently given by Chrobak and Larmore [CL2]. This issue of the computational resources required by server algorithms will be the subject of Chapter 5.

2.3 Other On-Line Problems

We now give a brief description of some other areas which have been studied from the perspective of competitive analysis, before turning to robot navigation, the other main area considered in this thesis. By now, many on-line problems have been studied; the following is not meant to be comprehensive.

As mentioned above, the surge of activity following [ST] led to some new results in on-line graph coloring, already a topic of interest in the 1970's. One basic question was the following: is there an on-line graph coloring algorithm with competitive ratio $o(n)$? Neither First Fit nor any of the other standard on-line coloring algorithms are better than $O(n)$-competitive, which is unfortunately also (to within constant factors) the competitive ratio of the algorithm which never re-uses a color. This question was answered in the affirmative in 1989 by Lovász, Saks, and Trotter [LST], who gave an on-line algorithm with performance ratio $O(\frac{n}{\log^* n})$. In terms of lower bounds, Halldorsson and Szegedy [HS] showed that for every on-line algorithm $\mathcal{A}$ and integer k, there is a graph with at most $k(2^k - 1)$ vertices and chromatic number k on which $\mathcal{A}$ will use $2^k - 1$ colors. Thus, no on-line algorithm can be better than $\Omega(\frac{n}{\log^2 n})$-competitive on all graphs. Related work on on-line graph coloring has been done by Irani [Ir] and Vishwanathan [Vi].

The problem of virtual circuit routing through a network is naturally on-line, and has been studied from that point of view in a number of recent papers. The model here is the following: one has a graph G which represents a high-speed network, and users at the nodes of the network who wish to transmit large amounts of data to each other. At each step, a pair of users requests that a *connection* be established between them. There are many variations, but in general a connection is specified by its two endpoints s and t, the amount of bandwidth requested, and the amount of time for which the connection must be in place. It is then the job of the on-line algorithm $\mathcal{A}$ to decide on a path from s to t on which the connection will be routed. Typically, $\mathcal{A}$ is trying to minimize the congestion in the network; thus, the cost incurred by $\mathcal{A}$ could be the maximum amount of bandwidth that passes through any link of G at any point in time.

The above description is essentially of the model considered in [AAFPW], and in [AAPW] when the duration of the connection is unknown but re-routing is allowed. An alternative, but similar, scenario is the following: the network does not have sufficient bandwidth to handle all requests; thus the on-line algorithm $\mathcal{A}$ must first decide whether to *accept* the connection, and if so, to route it through the network. This is the set-up considered in [AAP, ABFR]; a closely related problem is that of scheduling "intervals" on a line [LT]. One recurring theme in this work is that randomized algorithms often can achieve competitive ratios that are exponentially better than the lower bound for deterministic algorithms.

A somewhat related problem was proposed by Imase and Waxman [IW], though it has been suggested independently in a number of sources. This is the *on-line Steiner tree* problem: one has a metric space M, and n points in M which must be joined together by a minimum-cost Steiner tree. The sequence is presented on-line, and each new point must be immediately connected to the existing tree. The on-line/off-line distinction here sort of resembles an "urban growth" phenomenon — after the fact, it is much easier to find a reasonably short spanning network than when the points are appearing one at a time and must be hooked up immediately. [IW] shows that the greedy algorithm (when point x is requested, connect it to the existing network via the shortest possible path) is $O(\log n)$-competitive in any metric space; they also present a somewhat contrived metric space M in which no on-line algorithm can be better than $\Omega(\log n)$-competitive. Alon and Azar [AA] subsequently showed that in the Euclidean plane, no on-line Steiner tree algorithm can be better than $\Omega(\frac{\log n}{\log\log n})$-competitive, leaving a slight gap between the upper and lower bounds in this case.

2.4 Robot Navigation

Maze-solving is an old obsession. The interest in such problems can be seen in the labyrinths of mythology and medieval architecture; and, the bridges of Königsberg notwithstanding, it was one of the original motivations for study of graph theory. Maze-solving algorithms phrased in the terminology of graphs can be found in work of Tarry and Tremeaux that reaches back a century and more [Ore]. More recently, such questions have been addressed in the contexts of automata theory by Blum and Kozen [BK] and robotics by Lumelsky and Stepanov [LuS].

On-line navigation is concerned with questions somewhat more general than maze-solving: the environment will not always be nefarious, the goal not always to escape. Rather, we will be interested in a variety of tasks which must be performed in an environment for which the map, or some other crucial piece of information, is not known. Although the focus is theoretical, the motivation comes from the ever-growing field of autonomous mobile robots, and we should frequently ask whether the problems we consider have some bearing on the real-world issues of robotics.

Navigation in an unknown environment does not fit precisely into the framework described above, in that the mobile robot has a more interactive relationship with the "input" than in previous problems. Nevertheless, the problem is naturally on-line, in that the environment is presented in a piecemeal fashion, as the robot sees new parts of it, and the goal is (in general) to minimize the total distance traveled. The recent incorporation of navigation problems into the setting of competitive analysis comes mainly from the work of Baeza-Yates, Culberson, and Rawlins [BCR], and Papadimitriou and Yannakakis [PY].

[BCR] does not speak directly in terms of the competitive ratio, but its focus on the ratio of the robot's distance traveled to the length of the shortest path is clear enough. The questions it considers are variations on the theme of searching for an object at an unknown location in the plane or on a line. These include, for example, the problem of searching for a point on a line or a collection of lines, and the well-known "lost-at-sea" problem, in which one must determine the optimal search pattern for finding a line (the shore) in the plane (the ocean) [Be, Is].

A fundamental technique introduced in [BCR] is that of *spiral search* — an uncomplicated idea which has proved to be a valuable building block in numerous on-line algorithms. A good example of its use is in the problem of a robot searching for a *goal* which lies on one of n roads meeting at the origin. We will assume here that the robot is constrained to move on these roads and cannot use vision; however, it will recognize the goal when it comes to it. Moreover, to prevent certain pathologies, we will assume that the goal is at least one unit of distance away from the origin.

Clearly no on-line algorithm can be better than $\Omega(n)$-competitive for this problem: to construct a lower bound, we simulate the robot on a set of roads which does not contain a goal, and number the roads $r_1, \ldots, r_n$ according to when the robot first travels one unit of distance on each. We then simply place the goal one unit from the origin on road r_n. The robot will travel a distance of $2n - 1$, whereas the shortest path to the goal is 1. A simple form of the spiral search solution to this problem goes as follows: for $d = 1, 2, 4, \ldots, 2^j, \ldots$, explore each road out to a distance d and return to the origin. If the goal is found when $d = 1$, the robot has traveled at most $2n-1$ times the length of the shortest path. Otherwise, suppose the goal was found when $d = 2^j$ for some $j \geq 1$. Then the total distance traveled by the robot is at most

$$\sum_{i=0}^{j} n \cdot 2 \cdot 2^j < 2^{j+2} n.$$

Meanwhile, since the goal was not found when d was equal to 2^{j-1}, the length of the shortest path is at least 2^{j-1}. Thus, the robot travels at most $8n$ times the length of the shortest path. In [BCR] it is shown that by modifying these parameters a little, one achieve a competitive ratio of $2en - o(n)$, and that this is optimal up to low-order terms.

As is the case with most of the basic notions discussed here, one can find the spiral search technique being used implicitly in a wide range of previous papers; its wide applicability is obviously due to its simplicity — keep doubling until you succeed — and the relative ease of analyzing the performance guarantee one gets — the sum of all that you've done in previous phases is only a constant fraction of what you do in the current phase. Recently, it has been used in many of the robot search papers we will discuss below [BCR, PY, BRaSc, KRT, Kl], an abstract kind of navigation problem known as layered graph traversal [PY, FFKRRV], the design of hybrid algorithms [KMSY], and even the approximation of some NP-hard problems [BCCPRS, TWSY].

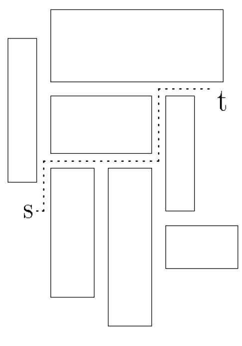


Figure 2-1: A rectangle packing

The work of Papadimitriou and Yannakakis followed [BCR] and deals with navigation problems that are somewhat less stylized. Specifically, they consider a robot with vision moving around in a plane filled with rectangular obstacles. The robot can move through any part of the plane not filled in by an obstacle, and there is always room to move between adjacent obstacles (i.e. they cannot be "stuck together" to make more complex shapes). By *vision*, both here and in what follows, we mean the following: if the robot is at x, it can see the point y if the line segment $\overline{xy}$ does not meet any obstacle. Finally, let $d(s,t)$ denote the length of the shortest obstacle-avoiding path between points s and t.

Based on the above definitions, we will say that a *rectangle packing* is a collection of axis-aligned rectangles, each of which has at least unit thickness, such that there is always just enough room for the robot to pass between neighboring rectangles. [PY] deals with shortest-path problems in this setting: the robot starts at a point s and knows the coordinates of a point t that it wants to reach. The point t is assumed to be in *free space* — i.e not in the middle of an obstacle. Let n denote the straight-line distance from s to t; note that this could be much less than $d(s,t)$. When all the obstacles in the packing are squares, [PY] gives an algorithm with competitive ratio $\frac{1}{3}\sqrt{26} \sim 1.70$, and shows that no on-line algorithm can be better than $\frac{3}{2}$-competitive. For the case of an arbitrary rectangle packing, no bounded competitive ratio is possible: [PY] shows a lower bound of $\Omega(\sqrt{n})$ on the best possible competitive ratio.

Blum, Raghavan, and Schieber [BRaSc] address the same type of shortest-paths problems and give an $O(\sqrt{n})$-competitive algorithm for the s-t path problem in a rectangle packing; this matches the lower bound of [PY] up to constant factors. [BRaSc] also introduced the elegant "room problem" — consider a rectangle packing inside a large $2n \times 2n$ square room (so there is space to move along the walls as well), such that the center of the room is in free space. The robot wants to start at one corner s of the room and reach the center t. The intriguing point here is that there is always a path from s to t of length $2n$ — simply start at t, move west as far as possible, then south as far as possible, then west ... and so on — and this construction is of course *on-line*. But can the robot starting at s somehow reverse this process and find a path to t of length $O(n)$? [BRaSc] left this as an open question, providing only an algorithm to generate a path of length $O(n2^{\sqrt{\log n}})$; the question was answered a year later by Bar-Eli, Berman, Fiat, and Yan [BBFY], who showed by a very complicated argument that no on-line algorithm can be guaranteed to find a path shorter than $\Omega(n \log n)$, and gave an algorithm with a performance guarantee matching this up to constant factors.

Less work has been done on geometric problems more in the spirit of [BCR] — that is, trying to find a short path to a goal t when neither the map of the environment nor the location of t is known. This will be the subject of Chapter 3.

Finally, there have been several papers on the related problem of *exploring* an unknown environment. Here, the goal is to traverse a path that sees all parts of the environment, both the obstacles and the free space. For the problem of exploring the interior (or exterior) of a simple rectilinear polygon, Deng, Kameda, and Papadimitriou [DKP] give a 2-competitive algorithm. The off-line version of this problem (what is the shortest path for exploring a given polygon P?) is perhaps better motivated as the "Shortest Watchman's Route Problem"; Chin and Ntafos [CN] give a polynomial-time algorithm for computing such a route, and some of their definitions are used in the algorithm of [DKP]. Betke, Rivest, and Singh [BRiSi] consider *compact* exploration algorithms: the robot always knows the shortest path back to the origin whenever it reaches a new point. Their main result is a compact exploration algorithm for rectangle packings, which travels at most a constant times the total perimeter length of all rectangles in the scene (this is a lower bound for the optimal solution if, for example, we assume that the robot has very limited vision).

For our purposes, exploration algorithms (on suitably restricted parts of the environment) will prove useful as subroutines in Chapters 3 and 4.

Chapter 3

Searching an Unknown Polygon

We are interested in geometric variants of the problem of searching with uncertainty. A natural problem in robotics is that of searching for a goal in an unknown polygonal region. For example, a robot with vision is placed at a starting point in a simple polygon, and it must traverse a path to some *target point* in the polygon. Both the location of the target and and the geometry of the polygon are unknown, but the robot will recognize the target when it sees it.

As discussed in the previous chapter, on-line search algorithms have generally been developed for situations in which the geometry is kept to a minimum — for example, the case of searching for a point on one of m concurrent rays [BCR, KRT, KMSY], a line in the plane, or a point in an integer lattice [BCR]. Moreover, the structure of the space to be searched is assumed to be known — only the location of the target is unknown. [BCR] writes, "...these problems are (very simple) models of searching in the real-world. It is very often the case that we do not know many of the parameters that are usually taken for granted in designing search algorithms."

Here, we provide an on-line algorithm for the general problem of searching for a target point at an unknown location in an unknown and arbitrary simple rectilinear polygon. Thus, the robot must adapt its search pattern as it sees more and more of the polygon. For a given rectilinear polygon P, we identify the number of *essential cuts* of P [CN, DKP] as a fundamental parameter in determining the best competitive ratio attainable in searching P — it is easy to cast the problem of searching m concurrent rays as a search problem in a polygon with m essential cuts, whence the [BCR] lower bound of $2em - o(m)$ on competitive ratio applies (e

is being used to denote the base of the natural logarithm here). Thus, the natural question is whether there is an algorithm which is $O(m)$-competitive for the problem of searching an *arbitrary* simple rectilinear polygon with m essential cuts. In Section 3.3, we present such an algorithm.

The algorithm is based on the problem — first considered in [DKP] — of exploring a simple rectilinear polygon P, starting from and returning to a fixed point s in P; we use an exploration algorithm iteratively as the search proceeds. We show in Section 3.2 how an adaptation of the technique in [DKP] gives a randomized algorithm which is 5/4-competitive in the L_1 norm when s is any point inside P.

Another direction in which one could investigate such search problems is to restrict the class of polygons and target points in such a way that a constant competitive ratio can be achieved for the search problem. Such an approach has been adopted by Klein in his work on *streets*. The papers [IK, K] introduce the term *street* to define a class of general (not necessarily rectilinear) polygons with two distinguished points s and t, such that the two st boundary chains are mutually weakly visible (see below for an elaboration on this definition). In [K], Klein gives a $1 + \frac{3}{2}\pi$-competitive ($\sim$ 5.71-competitive) algorithm for finding t from s in an unknown street P. However, the algorithm and its analysis are quite involved. In Section 3.1 we give a simple algorithm with a competitive ratio of at most $\sqrt{4+\sqrt{8}}$ ($\sim$ 2.61). Moreover when P is rectilinear, it achieves the optimal ratio of $\sqrt{2}$. We believe it would be interesting in general to find other natural classes of polygons that can be searched competitively.

Finally, a word about the distance metrics used in this chapter. The distance between two points as measured in the L_1 and L_2 (Euclidean) metrics differs at most by a factor of $\sqrt{2}$; thus, an algorithm which is c-competitive in L_1 is $c\sqrt{2}$-competitive in L_2. With the exception of the algorithm for traversing streets, which is analyzed directly in the Euclidean metric, we present our results in the conceptually neater framework of the L_1 metric. In view of the tight correspondence between L_1 and L_2, our final search algorithm is $O(m)$-competitive in both.

3.1 Traversing an Unknown Street

Let P be a simple polygon and s and t two distinguished points on the boundary. The removal of s and t would disconnect the boundary into two polygonal chains, L and R. We say that

P is a *street* [IK, K] if each point on the boundary of P can see some point on the opposite boundary chain. The goal is for a robot with vision to travel from s to t; neither the map of P nor the coordinates of t are known. The cost incurred by the robot is the length of the path it generates, and its competitive ratio is taken with respect to the length of the shortest s-t path in P; distances are measured in the Euclidean metric.

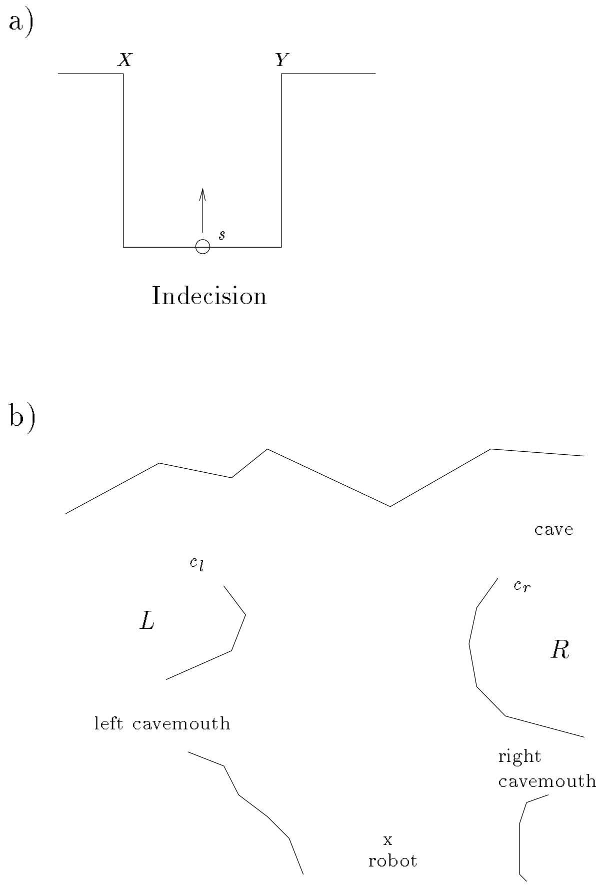


Figure 3-1: Streets and their views

Figure 3-1(a) can be completed to form a rectilinear street in which t could be just around the corner from either X or Y [K]. The robot will incur the best worst-case performance if it moves directly to segment $\overline{XY}$, then to t (it will see t when it reaches XY). This can be at most a factor of $\sqrt{2}$ longer than the shortest path. Curiously, this is the only known lower bound on the competitive ratio achievable for the problem. In [K], an algorithm with a competitive ratio of at most $1 + \frac{3}{2}\pi$ (~ 5.71) is presented. Below, we give an algorithm with competitive ratio at most $\sqrt{4+\sqrt{8}}$ (~ 2.61).

The example of Figure 3-1(a) is central to the proof technique we develop in this section, and it highlights a principle that will appear repeatedly in what follows — on-line algorithms hate making decisions. Specifically, it is useful in many navigation problems to adopt a strategy that preserves the robot's options for as long as possible. In the figure, suppose the robot at point s is moving towards segment $\overline{XY}$, but it has not yet decided whether it ultimately wants to visit point X or Y. Let us define a polygonal path to be *monotone* if the x- and y-coordinates of the points on the path change from their initial to final values monotonically. Then the key observation is that in the L_1 metric, any monotone path between two points is a shortest path, so the robot can defer its decision (X or Y) until it reaches segment $\overline{XY}$ and still have the option of traveling optimally to either point. The related fact for the Euclidean metric is that any monotone path between two points in the plane has length at most a factor of $\sqrt{2}$ times the straight-line distance between them. Thus, in this example, the robot can move to segment $\overline{XY}$ before making a decision and travel only $\sqrt{2}$ times too far (Euclidean distance) in the worst case.

Let P be a street, and assume that the robot is currently located at a point x inside P. The robot maintains an *extended view* of P; this consists of all points on the boundary of P that it has seen so far. The robot's extended view will typically look like the example of Figure 3-1(b). We define a *cave* C to be a connected chain of the boundary of P such that the robot has seen the endpoints of the chain but no other points of it. At some point p' on the robot's path, these two endpoints were on the same line of sight from p'; call the one closer to p' the "mouth" of C. Each cavemouth v is a reflex vertex of P; in the neighborhood of v, P lies either to the left or right of the ray $p'v$. We accordingly refer to v as being either a left or right cavemouth.

At any given point in time, it is useful to picture the "forward" direction for the robot as being along the positive y-axis; so the negative x-axis lies immediately to its left and the positive x-axis immediately to its right. In keeping with this terminology, if u and u' are two vectors, each with non-negative y coordinate, we will say that u is to the left of u' if it forms a smaller angle with the negative x-axis. Assume that t has not yet been seen, and the robot has maintained the invariant that the points in its extended view immediately to its left and right belong to L and R respectively. In view of these assumptions, we assemble some facts about extended views of a street before presenting the algorithm itself. The first is a standard fact about shortest paths inside any simple polygon.

Lemma 3.1 *If t is contained in a cave C, then the shortest x-t path touches the mouth of C.*

Lemma 3.2 *Let p be a point on boundary chain L (resp. R), and let Ψ be the boundary chain sp of P contained in L (R). If the robot moves from s to p, it will have seen every point on Ψ.*

Proof. Every point on boundary chain Ψ must be able to see some point on R; but all such lines of sight to R cross the robot's path from s to p. Thus the robot has seen every point on Ψ. ∎

Lemma 3.3 *If v is a left (right) cavemouth, it belongs to boundary chain L (R).*

Proof. Assume v is a left cavemouth, $v \in R$, and v was seen from point p'. The chain determined by a clockwise scan of the boundary from v to s (taken from point p') is entirely contained in R. Thus, if the robot were to walk directly from p' to v, it would have seen all of this chain, by Lemma 3.2. But since v is a cavemouth, it would not have seen any point on the boundary of P just around the corner from v, which belongs to this chain; this is a contradiction. ∎

Corollary 3.4 *In the extended view, all left cavemouths lie to the left of all right cavemouths.*

If the extended view contains any left cavemouths, we define c_l to be the rightmost one. The point c_r is defined analogously for right cavemouths. If both c_l and c_r are defined, the chain between the far endpoints of their respective caves must be completely visible in the extended view; otherwise, it would contain an additional left or right cavemouth. Combining this with Lemma 3.1 and the fact that t has not been seen,

Lemma 3.5 *The point t lies in the cave of either c_l or c_r. Consequently, the shortest path from x to t touches either c_l or c_r.*

Let $d(\cdot, \cdot)$ denote the length of the shortest path between two points in P. The shortest path from s to t, denoted by Γ, is a chain of line segments joined at reflex vertices of P. There is a natural order on the vertices of Γ, determined by traversing it from s to t. Our algorithm works iteratively, allowing the robot to move from a given vertex of Γ to a later one, with small "detour." The following theorem provides the main inductive step.

Theorem 3.6 *Assume that the robot is currently located at a vertex $x \in \Gamma$. Then it can move to a later vertex $x' \in \Gamma$ while traveling at most $(\sqrt{4+\sqrt{8}})d(x, x')$.*

Proof. We present an (on-line) algorithm for doing this. Based on the robot's extended view, there are four cases to consider. (See Figure 3-2.)

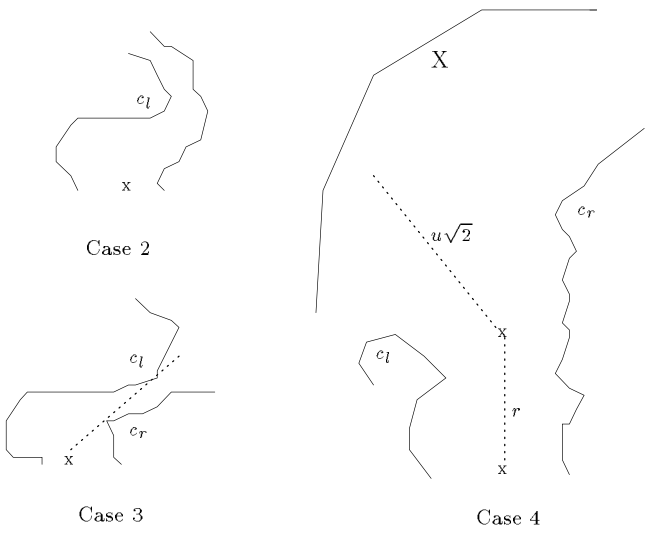


Figure 3-2: The algorithm at work

Case 1. If t is visible, the robot moves directly to t. The distance traveled is $d(x, t)$.

Case 2. If c_r (c_l) is not defined (there are no right (left) cavemouths), then by Lemma 3.5, Γ passes directly through c_l. Thus, the robot moves directly to c_l, following Γ the whole way.

Otherwise, both c_l and c_r are visible. The robot chooses a direction of motion such that c_l lies to its left and c_r lies to its right. We view this as a coordinate system in which the robot is the origin and it is moving in the direction of the positive y-axis; thus, c_l has negative x-coordinate and c_r has positive x-coordinate. The robot moves in this direction, updating its extended view and the points c_l, c_r, until one of of the above two cases applies, or one of the following two:

Case 3. The point c_l (or respectively c_r) "jumps" to the opposite side of the y-axis. At the moment when this happens, both c_l and c_r will lie on the same line of sight. If the robot moves in this direction until it hits the nearer one, it will once again be on a point $x' \in \Gamma$, having followed a path from x to x' that was monotone with respect to the chosen coordinate system. Thus, it has traveled no more than $\sqrt{2}$ times the distance from x to x' along Γ.

Case 4. If none of Cases 1, 2, or 3 applies, then there comes a point at which the robot's line of sight to c_l (c_r) is parallel to the x-axis. At the moment when this happens, the robot is "confused"; it can no longer follow a path guaranteed to be monotone to either c_l or c_r. However, the robot can adopt the following approach to return to the shortest path.

Let us translate the coordinate system so that the robot is again at the origin (so c_l now lies on the negative x-axis). Since t has not yet been seen, and since the second quadrant (i.e. $\{x \leq 0, \ y \geq 0\}$) is free of cavemouths, the robot can see all of the contiguous boundary chain X lying in this quadrant. Thus, X must be entirely contained in L or R; the robot can return to Γ once it discovers which of these cases holds. Define L_X to be the portion of boundary chain between c_l and the endpoint of X lying on the negative x-axis; define R_X to be the portion of boundary chain between c_r and the endpoint of X lying on the positive y-axis.

The robot begins moving in the direction of the vector $(-1, 1)$, updating its extended view and the points c_l, c_r, until it sees t or one of the following events occurs:

1. c_l has the same x-coordinate as the robot, or c_r has the same y-coordinate as the robot.

2. One of the chains L_X or R_X becomes completely visible.

If event (2) occurs first, then the robot will be able to move to the opposite cavemouth, thereby returning to Γ. Suppose event (1) occurs first; assume for the sake of concreteness that c_l has the same x-coordinate as the robot (the other case is strictly analogous). Then a point just around the corner from c_l can only see points lying on $X \cup L_X$; this implies that $X \subset R$, and so the robot can return to Γ by moving to c_l.

We must now bound how far the robot travels by implementing this strategy. Let r denote the distance traveled by the robot prior to becoming confused. First consider the case in which event (1) occurs, and c_r has the same y-coordinate as the robot. Suppose that the coordinates of c_r are (v, u); then the robot travels $r + u(1+\sqrt{2}) + v$, while we have $d(x, x') \geq \sqrt{(r+u)^2 + v^2}$. Thus, the worst-case ratio incurred by the robot is bounded by

$$\frac{r + u(1+\sqrt{2}) + v}{\sqrt{(r+u)^2 + v^2}}.$$

A somewhat lengthy but straightforward argument shows that this expression attains its maximum when $r = 0$ and $u = v(1+\sqrt{2})$, with a value of $\sqrt{4+\sqrt{8}}$.

Now suppose event (1) occurs, and c_l has the same x-coordinate as the robot. If the coordinates of c_l are $(-u, -v)$, then the robot travels $r + u(1+\sqrt{2}) + v$. Assume that when the robot first became confused, the coordinates of c_l were $(-u_1, 0)$ (recall that this point lies on the negative x-axis). Let $u_2 = u - u_1$; then we have

$$d(x, x') \geq \sqrt{r^2 + u_1^2} + \sqrt{v^2 + u_2^2}$$

In view of the simple inequality $\frac{a+b}{c+d} \leq \max(\frac{a}{c}, \frac{b}{d})$, the worst-case ratio is upper-bounded by the maximum value of

$$\frac{r + u_1(1+\sqrt{2})}{\sqrt{r^2 + u_1^2}}.$$

As above, this is maximized by taking $u_1 = r(1+\sqrt{2})$, with a value of $\sqrt{4+\sqrt{8}}$.

The case in which event (2) occurs first is similar. Suppose that all of L_X becomes visible and the goal has not been seen (the other case is analogous). Then we can set x' to be the current value of c_r. Suppose that the coordinates of c_r are (v, u); then since event (1) did not occur, the robot's y-coordinate is no more than u. Thus it can get to c_r having traveled at most $r + u(1+\sqrt{2}) + v$, while again $d(x, x') \geq \sqrt{(r+u)^2 + v^2}$.

Finally, we should note that it is possible for the robot's motion to be stopped by the boundary of P. As in Lemma 3.2, however, it will have seen the entirety of one of the caves associated with c_l or c_r by the time this happens, so it can return to Γ; the preceding analysis is not affected. ■

Corollary 3.7 *For any street P, repeatedly applying the above algorithm produces a path from s to t that is at most $\sqrt{4+\sqrt{8}}$ times as long as the shortest s-t path in P.*

In the case in which P is rectilinear, the above algorithm can be implemented so that the robot's direction of motion is always along one of the coordinate axes. Then the key observation is that Case 4 cannot occur in a rectilinear street (since all angles are right angles, either a point just around the corner from c_l could not see R, or a point just around the corner from c_r could not see L). Thus we can show

Theorem 3.8 *When P is rectilinear, the above algorithm is $\sqrt{2}$-competitive, and this is optimal.*

3.2 Exploring a Rectilinear Polygon

First, we present some basic definitions of [CN, DKP] on the structure of rectilinear polygons. In this and the remaining sections, distances will be measured in the L_1 metric.

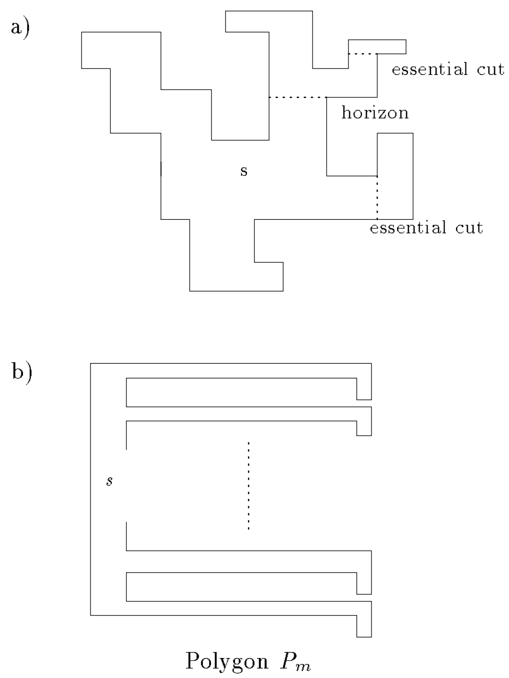


Figure 3-3: Some simple rectilinear polygons

Let P be a simple rectilinear polygon and s a distinguished point in P. An edge e of P is contained in a line ℓ; we say that an *extended edge* is a line segment $\hat{e} \subset \ell$ in P which shares one endpoint with e, and whose other endpoint is also on the boundary of P. Each edge e induces at most two extended edges. Also, note that an extended edge is either horizontal or vertical, depending on the orientation of its associated edge e. Call $\hat{e}$ a *horizon* if there is no path from s to e that does not cross $\hat{e}$. We can define a partial order on horizons as follows: if h and h' are horizons, then $h \preceq h'$ (h dominates h') if any path from s to h' must cross h, or if $h = h'$. The horizons of P which are maximal are called *essential cuts*; for an essential cut h, it is possible to start at s and follow a path which crosses every horizon except h. See Figure 3-3.

A special case of the following lemma is given in [DKP]; it is the underlying reason for the success of "greedy" exploration algorithms in rectilinear polygons.

Lemma 3.9 *Let $\alpha_1, \ldots, \alpha_n$ be a set of horizontal and vertical segments in P, and v a point in P. The (L_1) shortest path beginning at v and touching the α_i in order is generated by the greedy algorithm, which, from segment α_j, always chooses the shortest path to α_{j+1}.*

The problem we are considering here is that of traversing an *exploration route* in P: a path, starting and ending at s, such that every point of P can be seen from some point on the path. The robot does not know the map of P in advance. As noted in [CN, DKP] (see also the discussion above), a closed path through s has this property if and only if it touches all essential cuts in P. In [CN, DKP], it is observed that since any exploration route can be traversed (off-line) without self-crossings, the shortest exploration route will touch the essential cuts in clockwise order. Moreover, by Lemma 3.9, it will touch the cuts in this order using the greedy algorithm.

Consider the case in which the point s lies on the boundary of P, between the endpoints of essential cuts e and e'. The on-line algorithm given in [DKP] is essentially a greedy strategy which crosses each upcoming horizon as quickly as possible; it is shown in [DKP] that it will traverse the greedy path which touches the essential cuts in clockwise order, beginning with e. Consequently, this algorithm finds the *optimal* exploration route on-line; it is 1-competitive when s lies on the boundary of P.

When s does not lie on the boundary, the choice of which essential cut to start with becomes crucial, and the robot does not have enough information to make this choice.

Proposition 3.10 *No deterministic algorithm for exploring a simple rectilinear polygon can be better than 5/4-competitive.*

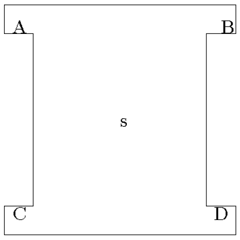


Figure 3-4: Lower bound construction

Proof. Consider Figure 3-4. All the long edges of the polygon P have length 2, the short edges have some length ε much less than 2, and s is at the center. A robot exploring P crosses either the upper or lower horizon first; assume the former case. At this point, it will see two tiny "caves" at points A and B, both of which must be visited. Assume that it visits A before visiting B or crossing the lower horizon (other cases are similar).

We now add an extra cave at C but not at D. Even if the robot now had the map of P, it would have to travel a distance of 8 to visit the caves at B and C and return to s. It has traveled a distance of 2 to reach A; thus its total distance is 10. On the other hand, the greedy exploration route which visits C first travels a distance of 8. ∎

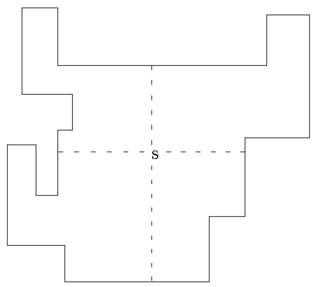
The algorithm given in [DKP] is 2-competitive when s is an arbitrary starting point in P, and this is the best known deterministic ratio. In the remainder of this section, we give a simple randomized algorithm whose performance matches the deterministic lower bound of Proposition 3.10.

Theorem 3.11 *There is* $5/4$*-competitive randomized exploration algorithm when* s *is an arbitrary starting point in* P*.*

Proof. Consider first the following construction. If the robot standing at s were to imagine a thin "needle" of boundary extending from the real boundary of P to s, it would then be on the boundary of this new polygon and could explore optimally. If we restrict ourselves to horizontal or vertical segments, then there are four possible needles that can be inserted in P.

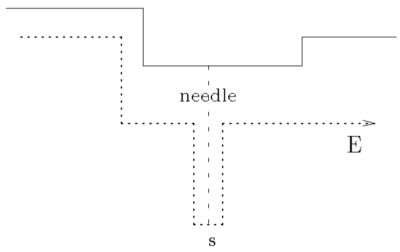
See Figure 3-5.

a)



Inserting needles in the polygon

b)



The exploration route E avoiding a needle

Figure 3-5: Polygon with needles

Let $L(\cdot)$ denote the length of a path in P, E denote the optimal exploration route in P, and P_i denote polygon P with the i^{th} needle inserted, $i = 1, 2, 3, 4$. Finally, we denote by T_i the (optimal) exploration route generated by the robot starting from s in P_i (s is on the boundary of each P_i). Of course, we are not really interested in the performance of T_i in P_i; we must show that T_i is also not far from optimal in the original polygon P. Set $d_i = L(T_i) - L(E)$.

We claim that $\sum_{i=1}^{4} d_i \leq L(E)$; the proof is as follows. Since E visits the essential cuts of P in clockwise order, it meets each needle in at most one point. E can be traversed so as to avoid the i^{th} needle (it takes a detour through s); let us denote this longer route by E_i. Since E_i is an exploration route for P_i and T_i is optimal in this polygon, we have $L(T_i) \leq L(E_i)$.

Let $d'_i = L(E_i) - L(E)$. Consider the four points at which E hits the needles (some of these points may be s); connect these by shortest paths to form a closed path $\tilde{T}$. $\tilde{T}$ is not necessarily an exploration route for P, but we have $L(\tilde{T}) \leq L(E)$ and $L(\tilde{T}) = \sum_{i=1}^{4} d'_i$. Since $d_i \leq d'_i$ for each i,

$$\sum_{i=1}^{4} d_i \leq \sum_{i=1}^{4} d_i' \leq L(E).$$

Thus, if the robot chooses any needle, the exploration route T_i it generates will have length at most $2L(E)$ (and simple examples show that, standing at s, there is no way to choose a needle guaranteeing a performance better than this). However, the expected value of the quantity d_i is bounded by $L(E)/4$, so if the robot chooses one of the four needles uniformly at random, the expected length of the exploration route it generates is at most $\frac{5}{4}L(E)$. ∎

3.3 Searching a Rectilinear Polygon

Using the exploration algorithm of the preceding section, we can give an $O(m)$-competitive algorithm for the general search problem described at the beginning of this chapter. A point t is hidden in a simple rectilinear polygon P with m essential cuts; however, the point t can be recognized when it is first seen. A robot starting at s, and without knowledge of the map of P, must travel to t; its distance traveled is compared to $d(s,t)$, the length of the shortest s-t path. As mentioned in the introduction, $O(m)$ is the best bound possible on the competitive ratio: if we take the polygon P_m of Figure 3-3, make the m "arms" extremely long, and introduce tiny bends to limit visibility, then the [BCR] lower bound for searching m concurrent rays applies — no deterministic algorithm can be better than $(2em - o(m))$-competitive. (An $\Omega(m)$ lower bound clearly holds for randomized algorithms as well, with a somewhat smaller constant.)

We first assemble some basic lemmas that will be useful in analyzing the exploration algorithm. Because P is a simple polygon, we have the following fact.

Lemma 3.12 *Let h and h' be horizons with the same orientation such that for some point v in P, every path from s to v must cross both h and h'. Then h and h' are comparable with respect to $\preceq$ ($h \preceq h'$ or $h' \preceq h$).*

Based on this lemma, we can represent the partial order $\preceq$ restricted to the horizontal segments by a directed tree T_h in which the root is the point s and the other nodes are horizontal horizons. For vertical horizons, there is the analogous representation as a tree T_v.

Lemma 3.13 *Each of T_h and T_v has at most m leaves.*

Proof. A leaf of T_h corresponds either to a horizontal essential cut, or a horizon h that dominates some vertical essential cut h'. In the latter case, no other leaf of T_h can dominate h', by Lemma 3.12, so h can be uniquely charged to h'. The same analysis holds for T_v, giving us the stated bound. ∎

Lemma 3.14 *Let v be a point in P such that s cannot see v. Then there is some horizon h in P that separates s from v, such that for any path from s to h, the point v can be seen from some point on this path. (I.e. no matter how the robot gets to h, it will have seen v.)*

Proof. By analogy with the construction for streets, consider the "view" of P from point v. Define a cave, as before, to be a connected chain of the boundary of P such that v can see the endpoints but no other points of the chain. Since s cannot see v, s must lie in some cave C. Let u be the cavemouth of C; then there is a horizon with endpoint u such that by the time the robot reaches this horizon, it will have crossed the ray $\vec{vu}$ and seen v. ∎

Lemma 3.15 *Suppose the robot is moving perpendicularly towards an extended edge e, and no other extended edge of the same orientation lies between the robot and e. Then the robot can see both endpoints of e.*

Proof. Suppose the robot could not see the right endpoint of e; let x be the rightmost point on e that it can see. The line of sight from the robot to x must meet a reflex vertex of P; it is easily verified that this is the endpoint of a parallel extended edge e' lying between the robot and e. ∎

Finally, the following lemma is the key to designing the search algorithm; it is also an interesting fact in itself. As before, let $d(u, v)$ denote the length of an L_1 shortest u-v path in P.

Lemma 3.16 *Let P be a simple polygon (not necessarily rectilinear), and consider a robot traversing some path in P. If points u and v are both visible from this path, then the robot can determine $d(u, v)$ without seeing the rest of P.*

Proof. In fact, it can compute a shortest path between u and v. Since we are dealing with the L_1 metric, the shortest u-v path in P will not generally be unique. However, some shortest u-v

path is polygonal (consists of a finite number of line segments). Consider the extended view of P that the robot has generated. By definition, the line segment joining the two endpoints of a cave in this view is completely contained in P; let us call it a "pseudo-edge." Let P' denote the truncated polygon whose boundary consists of the edges and pseudo-edges of the extended view of P. Thus, the robot has seen all of the boundary of P'.

We claim that there is a shortest u-v path in P that does not leave P'. The result will follow since the robot can obviously compute a shortest u-v path in P'. Let T be a polygonal shortest u-v path in P, which may enter a cave of the extended view, crossing pseudo-edge e at the point x. Since v lies in P', the path must re-enter P'; let us say that it next does so by crossing pseudo-edge e' at point y. Since P is a simple polygon, $e = e'$. Thus we can form a new u-v path which goes directly from x to y when T enters this cave; this operation does not increase the length. Proceeding in this way, we eliminate the (finitely many) places at which T enters a cave, producing a u-v path T' in P' with $L(T') \leq L(T)$. ∎

Theorem 3.17 *There is an algorithm for searching simple rectilinear polygons, which is $O(m)$-competitive on the class of such polygons with m essential cuts.*

Proof. For any $\delta > 0$, let P^δ denote the rectilinear polygon obtained by "truncating" P at horizons that are more than δ away from s. There is some small δ_0 such that all points within δ_0 of s are visible from s. By rescaling, we can assume that $\delta_0 = 1$. For fixed values of δ, the robot will simulate the algorithm of [DKP] in P^δ, as follows. The [DKP] algorithm has the property that at all times, the robot is moving along the coordinate axes, perpendicularly towards an extended edge, and it maintains this direction until it reaches the closest such extended edge e. By Lemma 3.15, it can see both of the endpoints of the edge e; thus, by Lemma 3.16, it can tell whether e lies in P^δ or not. If e lies in P^δ, the robot moves towards it, as in the [DKP] algorithm; otherwise, e is treated as a wall in the simulation. In this way, the exploration route generated by the simulation is the same as the route that the [DKP] algorithm generates in P^δ.

Initially, δ is set to 1. Whenever t is first seen, the robot moves directly to it; if the robot explores P^δ without seeing t, then δ is doubled and the next iteration begins.

Recall the partial orders T_h and T_v on the horizontal and vertical horizons of P. In P^δ, these naturally restrict to partial orders T_h^δ and T_v^δ, which are obtained simply by deleting all nodes corresponding to horizons more than δ away from s. Both T_h^δ and T_v^δ are still directed trees rooted at s, and by Lemma 3.13, each has at most m leaves. Since each essential cut of P^δ is a leaf of one of these trees, P^δ has at most $2m$ essential cuts. Lemma 3.14 implies that one way to explore P^δ would be to travel to each of its essential cuts individually; this would require a total distance of at most $2m(2\delta) = 4m\delta$. Since the robot is following a 2-competitive algorithm, it travels at most $8m\delta$ in exploring P^δ.

Given this, we can complete the proof as follows. If the robot sees t when $\delta = 1$, then it can travel to it optimally. Otherwise, assume it first sees t in P^δ, with $\delta = 2^k$. Since it did not see t when exploring $P^{(2^{k-1})}$, we have by Lemma 3.14 that $d(s,t) \geq 2^{k-1}$.

Now, we must bound how far the robot has traveled before reaching t. Until it sees t, it has traveled at most

$$\sum_{j=0}^{k} 8m2^j \leq 8m(2^{k+1}) \leq 32md(s,t).$$

Since $\delta = 2^k \leq 2d(s,t)$, it is easily verified that traveling to t can require no more than an additional $3d(s,t)$. Thus the robot's path is $O(m)$ times $d(s,t)$, as required. ∎

Chapter 4

The Robot Localization Problem

A fundamental task for an autonomous mobile robot is that of *localization* — determining its location in a known environment [Cox, TPBHSS, Wang]. This is also a problem well-known in everyday life, where it can be surprisingly easy to become lost even with a compass and some knowledge of the terrain — and where it is sometimes crucial whether or not the map one is using bears a little mark with the words, "You are here."

Thus we are dealing with a robot at an *unknown* location in an environment for which it *does* have a map. For our purposes, the environment $\mathcal{E}$ that we consider will generally be the interior of a large polygon filled with a finite number of polygonal obstacles. We also assume $\mathcal{E}$ is connected, so that there is an obstacle-avoiding path between any two points of $\mathcal{E}$. A robot — assumed throughout to possess vision, a map of $\mathcal{E}$, and knowledge of its orientation — "wakes up" at some point in $\mathcal{E}$; its goal is to determine where it is. Since we are assuming the robot knows its orientation, we will say, using somewhat non-standard terminology, that two regions are *congruent* if one can be obtained from the other by a translation (as opposed to a rigid motion).

The set of all points the robot can see from its current position forms a star-shaped *visibility polygon* P. If there is only one point in $\mathcal{E}$ from which the visibility polygon is congruent to P, then the robot can immediately determine uniquely where it is. Otherwise — if it is in a highly self-repeating environment such as a typical large building — there are a number of different places the robot could be. Guibas, Motwani, and Raghavan [GMR] considered the question of enumerating all possible locations for the robot, given a visibility polygon P, when the environment $\mathcal{E}$ is itself a simple polygon. They gave a "single-shot" algorithm which runs in time $O(|P||\mathcal{E}|)$, as well as more complicated data structures for answering "localization queries."

In a strong sense, though, enumerating the possible locations is only a first step towards solving the localization problem. For if the environment is connected and robot begins moving around in $\mathcal{E}$, building up larger and larger partial maps $P' \supset P$ of its "neighborhood," there will come a time when there is only one point in all of $\mathcal{E}$ at which one can place a partial map congruent to P'; the robot has then determined exactly where it is. (We will say that it has *localized.*)

One way to see that the robot will eventually localize is to note the straightforward fact that if it explores all of $\mathcal{E}$, then it can determine where it started from. But this is a crude solution; one would like to design an algorithm which causes the robot to travel as little as possible. From this point of view, a good localization algorithm is constantly adapting its search based on the set of possible locations of the robot, and their position in the map. We can therefore phrase the following question: how should the robot move in $\mathcal{E}$ so as to determine its location efficiently?

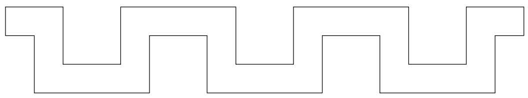


Figure 4-1: The infinite fence problem

Although the question is quite precise — we can choose to measure efficiency in terms of total distance traveled — it is not immediately clear that it can have an interesting answer. Indeed, if we consider the polygon of Figure 4-1, a robot starting from the middle has no choice but to walk to one of the ends before it can determine where it is; and this is only a factor of two better than the trivial approach of exploring the whole environment. So in what sense can we say that one algorithm is significantly more efficient than another?

We argue here that this is an ideal setting in which to use the competitive ratio. Specifically, in the polygon of Figure 4-1, the key point to observe is that the robot could not have done any better. Even if we had an omniscient robot that knew where it was, it would have to travel half the diameter of the polygon before being able to *prove* that it started from the middle. In this sense, localization becomes an on-line problem: we define a *localizing path* to be any path followed by a robot, at the end of which it can determine uniquely where it is, and compare the distance an on-line robot travels before localizing to the length of the shortest possible localizing path. The ratio of these two distances is our performance measure for a localization algorithm; the worst-case value of this ratio, over all starting points in $\mathcal{E}$, is its competitive ratio.

We find the search for good localization algorithms interesting for two fundamental reasons:

1. Localization is a problem that arises in numerous practical robotics situations.

2. The use of the map in helping to guide the robot's search introduces some complications that are not generally found in on-line navigation problems. In this case, it means that straightforward on-line techniques such as spiral search are no longer asymptotically optimal.

Applications in Robotics

The localization problem has been considered in a wide variety of contexts in the robotics literature. One application is in the design of robot vehicles that must perform a certain task repeatedly in the same environment. Here, localization is used to determine the starting location at the beginning of the task, and to maintain positioning information over time [Cox, Dr, Wang]. A similar use of localization is in analyzing aerial photographs to determine the location from which they were taken [YD].

Another major situation in which localization is used is in the design of autonomous exploration vehicles, such as the current prototypes for Mars rovers [MAWM, SN]. For example, [MAWM] discusses strategies by which a mobile robot can determine its location after a short period of "reconnaissance": in the model considered there, a rough global map of the Martian terrain is known; the exploring robot relays local information back to Earth, where an obstacle-avoiding path is then planned.

In a related vein, [TPBHSS] draws a distinction between localization algorithms for "update" problems, in which there is some rough estimate of the robot's starting location, and "drop-off" problems, in which there is no initial information. In this terminology, we will be considering the latter type of problem. Again, [TPBHSS] describes the role of "reconnaissance" in the localization problem, without providing any concrete algorithms.

Theoretical Background

The only previous theoretical treatment of the localization problem is the paper of Guibas, Motwani, and Raghavan [GMR], and as mentioned above, it deals only with the "static" version of the question. There has been no previous work on this problem from the perspective of competitive analysis, though there is a straightforward (and non-optimal) algorithm which is not difficult to describe; we will discuss this shortly.

In what follows, we deal with fairly simple types of environments in order to emphasize the combinatorial aspects of the problem rather than the geometric ones. Thus, we consider the following two types of environments.

- Bounded-degree trees embedded in $\mathbf{R}^d$. That is, the vertices are realized by points in $\mathbf{R}^d$ and the edges by line segments, and the robot is constrained to move on the edges and vertices. If T is an embedded tree, we will use n to denote the number of leaves it has.

- Rectangle packings in the plane, comprised of n rectangles [PY, BRaSc, BBFY, BRiSi]. Recall from Chapter 2 that we assume all rectangles have at least unit thickness, and that there is always just enough room for the robot to move between neighboring rectangles.

Arguably the most natural way to design a localization algorithm for either of these environments is the spiral search technique of Baeza-Yates, Culberson, and Rawlins [BCR]. For our purposes here, spiral search could be implemented by having the robot iteratively explore all points of the environment within distance $1, 2, 4, \ldots, 2^j, \ldots$, until it knows where it is; the resulting algorithm will be $O(n)$-competitive.

In navigation problems for which no information about the map is known (such as those of the previous chapters), it is frequently not difficult to show that spiral search is optimal up to constant factors. Indeed a great deal of work is often done to determine what these constant factors are [BCR, KRT, KMSY]. But the fact remains that no algorithm can be better than $\Omega(n)$-competitive for many of these problems. In some sense, this is not surprising; when the robot knows nothing at all, it is difficult to do much besides a brute-force search out to larger and larger radii.

What we find interesting about the localization problem is that knowledge of the map enables the robot to begin focusing its search as it sees more and more of its surroundings; in particular, this will lead to a localization algorithm that is $o(n)$-competitive,[1] improving asymptotically on spiral search. At a more general level, we are interested in on-line navigation problems in which, as is common in real applications, the robot has some limited information about its environment. Such problems tend to contain interesting structure that can be exploited when designing algorithms, and often provide insight into the value of a map in performing navigation tasks. Another example of this is the k-trip shortest-path problem considered by Blum and Chalasani [BC]. Here the robot wishes to make k trips between points s and t while minimizing the *average* time per trip; thus it can make use of partial maps of the scene on later trips. For a further perspective on the value of different types of information in performing navigation tasks, see the work of Donald on "information invariants" [Don].

The localization algorithm we present is quite natural and simple to state; the difficulty lies in analyzing the competitive ratio. The algorithm performs an initial period of spiral search on a local area which is sufficiently restricted to keep the competitive ratio from getting too large. At some point during this search, the robot is able to identify one or more "critical directions" in the environment; by searching only in these directions, the number of possible locations can be eliminated much more quickly. The algorithm then performs a final "clean-up" stage, in which the remaining possibilities are eliminated in an iterative fashion. In the case of trees, this algorithm is $O(n^{2/3})$-competitive. The analysis for rectangular obstacles appears to be quite a bit more complicated; however, we are able to show a competitive ratio of $O(n\sqrt{\frac{\log\log n}{\log n}}) = o(n)$. In contrast, the strongest lower bound we can show is $\Omega(\sqrt{n})$, in both types of environments. Closing this gap remains an interesting open question.

In independent work, Dudek, Romanik, and Whitesides [DRW] give a geometric implementation of the "shortest-distinguishing-paths" algorithm — which appears as Step 3 in the main algorithm of this chapter — when the environment is a simple polygon. Their algorithm is $O(k)$-competitive when the robot initially cannot distinguish among k possible starting points

[1] Recall that $f(n)$ is $o(g(n))$ when $\lim_{n\to\infty} f(n)/g(n) = 0$.

(this number may be as large as $\Omega(n)$, where n is the number of vertices of the polygon).

4.1 Lower Bounds and Other Examples

We give two examples of trees embedded in the plane, both of which show that no localization algorithm can be better than $\Omega(\sqrt{n})$-competitive on trees. In this and the following section, we assume that a robot on a tree cannot make use of vision — all it knows is the orientation of all edges incident to the vertex it currently occupies. Note that this can be simulated in a simple rectilinear polygon by introducing tiny bends everywhere to limit visibility; thus, any lower bound for trees immediately carries over to the class of simple polygons.

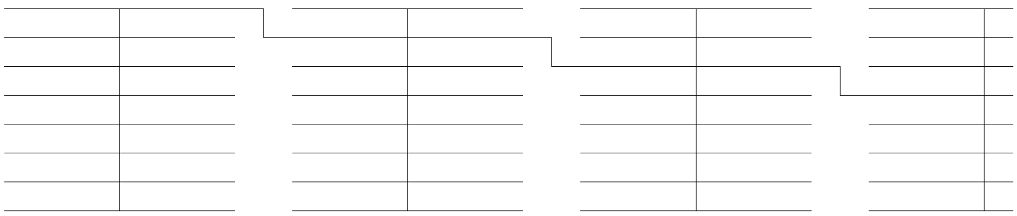


Figure 4-2: The first lower bound

By a *branch-vertex*, we mean a vertex with degree at least 3. The tree T in Figure 4-2 consists of p blocks, each of which has p branch-vertices and long "passages" to its neighboring blocks. Thus the number of leaves in T is $n = 2(2p-1) + (p-2)(2p-2) = 2p^2 - 2p + 2$. The robot cannot localize until it has found one of the long passages; by placing it in the block whose passage corresponds to one of the branches it will search last, we can force the robot to travel $\Omega(p) = \Omega(\sqrt{n})$ times too far. (In fact, there is an algorithm which is $O(\sqrt{n})$-competitive on this tree; this is not difficult to see.)

We now turn to a second $\Omega(\sqrt{n})$ lower bound for trees, which is somewhat more subtle to analyze and also has the advantage of extending to rectangular obstacles. The tree T_h is shown in Figure 4-3; it consists of a path Π of length n, with a path growing north from each vertex. All paths but the middle one have length h; the middle one has length $h+1$ (the value of h will be fixed later). Observe that T_h indeed has n leaves; and the key observation is that the robot cannot localize until it has reached one end of Π, or traversed the middle path.

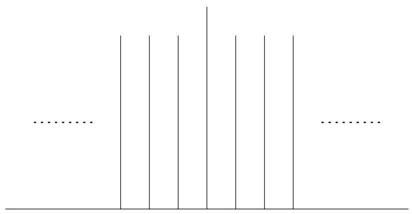


Figure 4-3: The second lower bound

First we argue that if $h = \lfloor\sqrt{n}\rfloor$, then no algorithm on T_h can be better than $\Omega(\sqrt{n})$-competitive. Let $v_1, \ldots, v_h$ be the vertices of Π that lie between h and $2h$ steps to the west of the entrance to the middle path. We place the robot at one of these vertices, such that it will traverse all the h other possible north-leading paths before finding the middle one. Thus, it will travel at least $h^2 \geq n - 2\sqrt{n}$ before finding the end of the middle path; to reach either end of Π, it must travel at least $n/2 - 2\sqrt{n}$. Either way this is a distance of $\Omega(n)$. Meanwhile, the off-line adversary need walk a distance of at most $3\sqrt{n}$ to reach the end of the middle path.

Conversely, however, one cannot improve this lower bound by varying the value of h. Assume the robot starts somewhere on the path Π, a distance d from the entrance to the middle path. (if it starts elsewhere, the arguments are essentially the same). If $h < \sqrt{n}$, then the robot can apply the two-way spiral search algorithm, traveling all the way up each new path it encounters. This algorithm is $O(h)$-competitive. And if $h \geq \sqrt{n}$, then the adversary must travel at least a distance of $\sqrt{n}$ to localize, so the robot can ignore all the north-leading paths and simply apply the two-way spiral search algorithm to the path Π until the nearer end is reached, traveling a distance $O(n)$.

This latter case suggests some of the difficulty inherent in obtaining a $o(n)$-competitive algorithm — when $h = n$, the full spiral search algorithm is $\Omega(n)$-competitive; it is necessary to identify the path Π as somehow being more important than the profusion of paths leading north. We will see a general technique for doing this in the next section.

In closing this section, we simply note that it is easy to construct the second lower bound out of $O(n)$ rectangles; see Figure 4-4. We use very long rectangles growing north and south so that it will take the robot too long to try completely leaving the middle path Π. Each of the paths growing north out of Π is now simulated by two of these very long rectangles with a

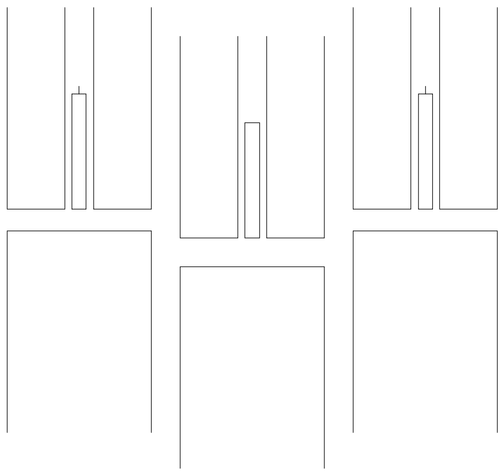


Figure 4-4: Making the second lower bound work for rectangles

shorter one in between; also Π now bends slightly so that the robot cannot see all the way down it. Thus no algorithm can have a performance guarantee better than $\Omega(\sqrt{n})$ for environments of n rectangular obstacles.

4.2 The Algorithm for Trees

Recall that by a geometric tree T we mean a pair (V, E), where V is a finite point set in $\mathbf{R}^d$ and E is a set of line segments whose endpoints all lie in V. The segments of E intersect only at points in V, and they do not induce any cycles.

As before, vertices with degree greater than 2 in T will be called *branch-vertices*. It will turn out that the degree-2 vertices of T are largely unimportant in the algorithm; thus we change our definition of T to an equivalent one without such vertices. Specifically, we will say that all the vertices of a geometric tree T are either leaves or branch-vertices, and edges are now polygonal paths between the vertices. Moreover, we assume for the sake of simplicity that T has bounded degree; namely, for some absolute constant Δ, at most Δ segments in E are incident to any given vertex in V. So we again change notation slightly by saying that T has n branch-vertices, and consequently has at most $(\Delta - 2)n + 2 \leq \Delta n$ leaves (and at least $n + 2$).

If $U \subset V$, we use $T(U)$ to denote the subtree induced by U. When $U = \{x, y\}$, this is simply the path from x to y; we denote its length (as a polygonal path under the Euclidean distance) by $d_T(x.y)$. Finally, recall that the robot is constrained to move on the vertices and edges of T, and can make no use of vision other than to know the orientation of all edges incident to its current location.

Consider a geometric tree T with n branch-vertices. We wish to prove the following.

Theorem 4.1 *There is an algorithm which is $O(n^{2/3})$-competitive for the localization problem on geometric trees.*

To prevent various pathologies, we assume that all the points in V have rational coordinates, and that the minimum length of any edge in E is 1. (These assumptions can be avoided at the cost of more cumbersome definitions below.)

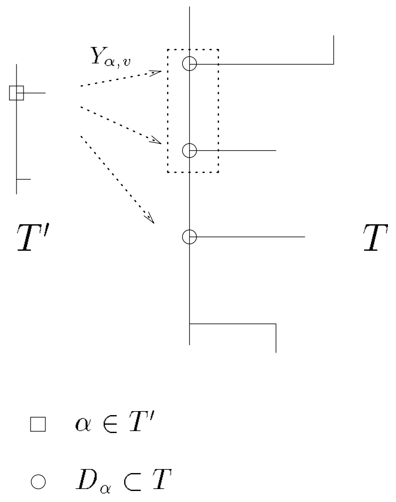


Figure 4-5: Lining up T' with T

We can assume without loss of generality that the robot begins at one of the *vertices* of T (i.e. at a leaf or a branch-vertex), rather than in the middle of one of the edges. This is simply because the robot can initially perform two-way spiral search to reach the closest vertex, traveling no more than 9 times too far. Moreover, we can then assume that it in fact begins at a branch-vertex, since its only choice at a leaf is to move along the incident edge.

At all times, the robot maintains a *search region*, some geometric tree T' which consists of everything it has seen so far. This means that it is in a part of T which locally looks like T';

thus the basic computation the robot will be performing as it explores is that of "lining up" its current copy of T' with various parts of T, in the natural way. The robot maintains its current location α on the map of T', and a set D_α of *possible* locations v that this might correspond to in T. (To keep the notation clear, we will use Greek letters to denote vertices in T'.) If $v \in D_\alpha$, then we will use $Y_{\alpha,v}(T')$ to denote the subtree of T induced by rigidly placing T' on T so that α is mapped to v. See Figure 4-5.

Of course, the robot always does have a "genuine" location in T; it simply does not know what this is (until it has localized). Notationally, it is sometimes useful to refer to this unknown location: when the robot is at $\alpha \in T'$, we will denote its true position in T by $Z(\alpha)$. (Note that $Z(\alpha) \in D_\alpha$.)

The following fact is immediate but will be used frequently.

Lemma 4.2 *For all α, β in T', D_β is equal to the set D_α translated by the vector from α to β. In particular, $|D_\alpha| = |D_\beta|$.*

As the robot performs its exploration, it remembers the branch-vertex in T' at which it first woke up; we will denote this vertex γ_0. In the course of the localization algorithm, the robot maintains four principal quantities of the search region.

1. As we will see below, T' is initially constructed using spiral search. Thus we maintain the current "search radius" $r(T')$; this is the distance out to which depth-first search is currently being performed, from the initial location γ_0 in T'.

2. The common value of $|D_\alpha|$ (as in Lemma 4.2) will be denoted $p(T')$; this is simply the number of possible placements of T' in T.

3. The number of branch-vertices of T' will be denoted $b(T')$.

4. The quantity $w(T')$ is defined to be

$$\max_{\alpha \in T', v \in D_\alpha} |D_\alpha \cap Y_{\alpha,v}(T')|.$$

   That is, if we pick α to be the "origin" of T', then we can place it on v so that it covers $w(T')$ other origins.

The algorithm consists of three main steps, which are controlled by the following global structure:

```
Initially b(T') = 1, p(T') = O(n), w(T') = 1
While b(T') ≤ p(T') and b(T') ≤ n/w(T')
  Execute Step 1
If p(T') < b(T') then
  Execute Step 3
Else execute Step 2 followed by Step 3
```

In the remainder of this section, we will describe each of the steps individually, then analyze the competitive ratio of the resulting algorithm.

Step 1: Restricted Spiral Search

The robot first wakes up at some initial location $\gamma_0 \in T'$ and begins performing spiral search [BCR]. This can be described as follows: the robot starts at γ_0 and performs successive depth-first searches so as to see all points within distance 2^j of $Z(\gamma_0)$, for $j = 0, 1, 2, \ldots$.

The fundamental fact about spiral search is the following.

Lemma 4.3 *At the end of Step 1, the robot has traveled no more than $8\Delta b(T')$ times the length of the optimal solution.*

Proof. Suppose the final search radius was $r = 2^j$. So the total distance traveled by the robot is at most

$$\sum_{i=0}^{j} 2\Delta b(T') \cdot 2^i \leq 2^{j+2}\Delta b(T').$$

Since Step 1 did not terminate when the search radius was equal to 2^{j-1}, any localizing path must travel a distance of at least 2^{j-1} away from γ_0. The bound follows. ∎

Step 2: Extending the Critical Path

If P is a directed polygonal path and $k \geq 1$, we use P^k to denote the path formed by joining together k copies of P in succession. We use P^{-1} to denote P with the edges presented in the reverse order. Let $U = \{u_1, \ldots, u_k\}$ be a subset of the vertices of T. We will say that U

(a)

(b)

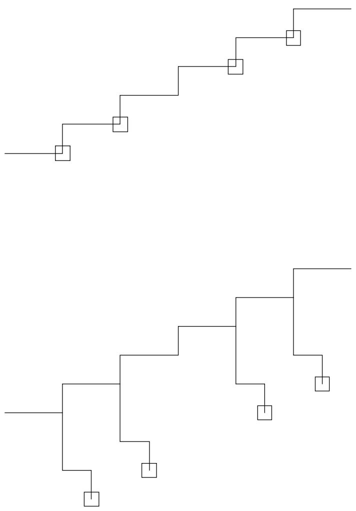


Figure 4-6: (a) A periodic path. (b) A comb tree.

induces a *periodic path* if $T(U)$ is a simple path and there is some polygonal path P such that $T(u_i, u_{i+1}) = P^{m_i}$ for natural numbers $m_1, \ldots, m_{k-1}$. We will say that U induces a *comb tree* if $T(U)$ consists of a periodic path on vertices $\{v_1, \ldots, v_k\}$ distinct from U (the *base*), together with disjoint paths $T(u_i, v_i)$ (the *teeth*), all of which are mutually congruent; the vertex v_i will be called the *support point* of u_i. (See Figure 4-6.)

Recall that at the beginning of Step 2, we have an "origin" α of T' and a placement $Y_{\alpha,v}$ of T' which covers at least $w(T')$ other origins in T. We will use W to denote this set

$$D_\alpha \cap Y_{\alpha,v}(T')$$

of covered origins.

Lemma 4.4 *W induces either a periodic path or a comb tree in T (and hence also in T').*

Proof. The proof is based on the following two claims, whose proofs are given in the Appendix of this chapter.

Claim 4.5 *Let T be a geometric tree, $r, x, y, z \in V$. Suppose that $T(r, x)$ is congruent to $T(y, z)$, and $T(r, y)$ is congruent to $T(x, z)$. Then $T(r, x, y, z)$ is either a periodic path or a comb tree.*

Claim 4.6 *Let T be a geometric tree and U a subset of the vertices. Suppose that there is some $x \in U$ such that for all other $y, z \in U$, $T(x, y, z)$ is either a periodic path or a comb tree. Then $T(U)$ is a periodic path or a comb tree.*

Using these, we prove the lemma as follows. Recalling that $W = D_\alpha \cap Y_{\alpha,v}(T')$, let us suppose that $W = \{w_1, \ldots, w_k\}$, with $w_1 = v$. Consider any other w_i, w_j with $1 < i < j \leq k$. Since these belong to D_α, there is a path congruent to $T(w_1, w_i)$ emanating from w_j, and a path congruent to $T(w_1, w_j)$ emanating from w_i. Note that these two paths have a common endpoint (at the point $w_1 + (w_i - w_1) + (w_j - w_1)$). Let u denote this common endpoint; then w_1, w_i, w_j, u satisfy the hypotheses of Claim 4.5. This in turn shows that W and the distinguished point w_1 satisfy the hypotheses of Claim 4.6, and hence $T(W)$ is either a periodic path or a comb tree. ∎

But suppose that $T(W)$ is in fact a comb tree. Then by Lemma 4.2, there is a vertex $\alpha' \in T'$ such that $D_{\alpha'}$ contains the set W' of support points of $T(W)$ (which form a periodic path). In fact, if we let v' denote the support point of v, we see that $W' \subset D_{\alpha'} \cap Y_{\alpha',v'}$, so

$$|D_{\alpha'} \cap Y_{\alpha',v'}| \geq |W| \geq w(T').$$

Thus, by using α' instead of α, we obtain a set W' (satisfying $|W'| \geq w(T')$) which induces a periodic path. We state this as the following extension of the previous lemma.

Lemma 4.7 *W can be chosen so that it induces a periodic path in T (and in T').*

Suppose $T(W) = T'(W) = P^m$ is such a periodic path, with origin α. The robot moves to α; it is now standing somewhere in the middle of a long periodic path, and can thus follow successive copies of P by moving in one direction (the *forward* direction), and successive copies of P^{-1} by moving in the other (the *backward* direction). The robot moves in each of these directions until it determines the largest i and j for which it is possible to traverse P^i in the forward direction and P^{-j} in the backward direction, starting from α.

Locating α and v, and traversing P^i and P^{-j}, constitutes Step 2. Note the following facts.

Lemma 4.8 *In performing Step 2, the robot travels no more than $4 + \frac{8n}{w(T')-1}$ times the length of the optimal solution.*

Proof. Moving to α costs at most $2r(T')$. T' contains P^m as a path, and $m \geq w(T')$, so the length of P (and hence P^{-1}) is at most $\frac{2r(T')}{w(T')-1}$. Also, each copy of P uses up an additional branch-vertex of T, so the robot will traverse at most n copies of P and P^{-1}. Since each copy is traversed twice (the robot returns to α), it travels at most $2r(T') + \frac{4nr(T')}{w(T')-1}$. On the other hand, the length of the optimal solution is at least $r(T')/2$, as argued above. ∎

Lemma 4.9 *At the end of Step 2,* $p(T') \leq 2n/w'(T')$.

Proof. For $v \in D_\alpha$ and u any other vertex of T, we will say that u is P-covered (resp. P^{-1}-covered) by v if by starting at v and following successive copies of P (resp. P^{-1}), the robot can reach u. By considering the path P^m in $Y_{\alpha,v}$, we see that the total number of vertices that are P-covered or P^{-1}-covered by each $v \in D_\alpha$ is at least $w(T')$ (counting v itself). Reversing the names of P and P^{-1} if necessary, we can assume that the average number of vertices P-covered by a vertex $v \in D_\alpha$ is at least $w(T')/2$.

On the other hand, we claim that no vertex of D_α P-covers any other. This is simply because it is well-defined for each $v \in D_\alpha$ how far one can move along T following the edge sequence of P^n; thus if v P-covers v', the robot can eliminate at least one of v or v' from its set of possible locations.

This in turn implies that no vertex $u \in T$ is P-covered by more than one member of D_α (if it were covered by two, the one farther from u would cover the one closer to u). Since the average number of vertices covered by a vertex $v \in D_\alpha$ is at least $w(T')/2$, and each vertex is covered at most once,

$$|D_\alpha| = p(T') \leq \frac{2n}{w(T')}.$$

∎

Step 3: Cleaning Up

Once $p(T')$ has become sufficiently small, the robot can finish its task by brute force. Specifically, assume that it is currently located at an origin α in T'. For each pair of vertices $v, v' \in D_\alpha$, define their shortest *distinguishing path* to be the shortest path Q such that it is possible to traverse Q starting from v but not from v' (or vice versa). (This is the least one has to travel to tell v from v'.)

In Step 3, the robot iteratively applies the following strategy. Over all $v, v' \in D_\alpha$, it chooses the pair with the distinguishing path of minimum length. By following this path, the robot will be able to eliminate either v or v' from D_α, and perhaps both. Meanwhile, the optimal off-line algorithm must travel at least this far, since there is no way to eliminate even a single vertex from D_α otherwise. The robot then returns to α and begins the next iteration. As there are at most $p(T') - 1$ such iterations, we have proved

Lemma 4.10 *In Step 3, the robot travels no more than $2p(T') - 2$ times the length of the optimal solution.*

The Global Structure

Finally, we give an absolute bound on the competitive ratio, using the lemmas above. Note first of all that the tree T_h of Figure 4-3, with $h = n$, shows that the algorithm which simply applies Step 1 until $p(T') \leq b(T')$ and then switches to Step 3 is no better than $O(n)$-competitive. But introducing the option of Step 2 prevents the initial period of spiral search from going on for too long. The crucial fact is the following.

Lemma 4.11 *By the time $b(T')$ exceeds $n^{2/3}$, the robot will have stopped executing Step 1.*

Proof. For $\alpha \in T'$ and $v \in D_\alpha$, let $w_{\alpha,v}(T')$ denote the cardinality of $D_\alpha \cap Y_{\alpha,v}(T')$ — that is, the number of origins in D_α covered if T' is placed so that α corresponds to $v \in T$. To prove the lemma, it is sufficient to show that if both $b(T')$ and $p(T')$ are greater than $n^{2/3}$, then some $w_{\alpha,v}(T')$ is at least $n^{1/3}$.

For a given $v \in T$, let

$$E_v = \{\alpha : v \in D_\alpha\}$$

and $e_v = |E_v|$. Thus,

$$\sum_{v \in V} e_v = b(T')p(T').$$

Now let us compute the sum S of $w_{\alpha,v}(T')$ over all pairs (α, v) such that $v \in D_\alpha$. Fix such a pair (α, v). First, it contributes once to $w_{\alpha,v}(T')$. Also, for every other $\beta \in E_v$, there is some $u \in T$ such that v corresponds to β in $Y_{\alpha,u}(T')$; thus, the pair (α, v) contributes once to

$w_{\alpha,u}(T')$. Thus (α, v) contributes a total of e_v to the sum S. Summing over all pairs, we obtain

$$\sum_{\alpha,v} w_{\alpha,v}(T') = \sum_v e_v^2.$$

Since T has n branch-vertices, this value is minimized by setting each e_v equal to

$$\frac{1}{n}\sum_v e_v = \frac{1}{n}b(T')p(T')$$

and thus

$$\sum_{\alpha,v} w_{\alpha,v} = \sum_v e_v^2 \geq n \cdot \frac{1}{n^2}b(T')^2p(T')^2 = \frac{1}{n}b(T')^2p(T')^2.$$

By the pigeonhole principle, applied to the $b(T')$ choices of α and the $p(T')$ choices for $v \in \alpha$, there is some pair v, α for which

$$w_{\alpha,v} \geq \frac{1}{n}b(T')p(T') \geq n^{1/3}.$$

as desired. ∎

First suppose the robot goes directly from Step 1 to Step 3. When this transition happens, $b(T') \leq n^{2/3}$ and $p(T') \leq n^{2/3}$, so by Lemmas 4.3 and 4.10, it travels at most $O(n^{2/3})$ times the length of the optimal solution.

Otherwise, the robot goes from Step 1 to Step 2, the transition occurrring when $b(T')$ and $n/w(T')$ are at most $n^{2/3}$. Thus by Lemmas 4.3 and 4.8, it travels at most $O(n^{2/3})$ times too far in Steps 1 and 2. Now, Lemma 4.9 implies that it will begin Step 3 with $p(T') \leq 2n^{2/3}$, so by Lemma 4.10, it travels only $O(n^{2/3})$ times too far in Step 3 as well. As these are the only two cases, this completes the proof of Theorem 4.1.

4.3 The Algorithm for Rectangles

Again, to keep complications related to visibility to a minimum, we work with a rectangle packing [BRaSc, BBFY, BRiSi], as defined earlier. By a *vertex* of the environment, we will mean a corner of some rectangle. So in the spirit of the previous section, one could picture a planar graph embedded in the two-dimensional integer grid, all of whose bounded faces are rectangles.

The algorithm for the case of rectangular obstacles is very similar to the one for trees; the main difference is the lack of an analogue to Lemma 4.4 to provide the robot with an obvious critical path to explore. As a result, the transition from Step 1 to Step 2 cannot happen as early as in the algorithm of Section 4.2; rather, the robot waits until the spiral-search branching factor becomes too large, and then begins exploring *several* critical paths in succession before beginning Step 3. We elaborate below.

Let $\lambda(n) = \sqrt{\frac{\log n}{\log\log n}}$. The set of n rectangles will be denoted R, and the current search region R'. $b(R')$ will now simply be a measure of the number of rectangles in R' (including partial rectangles). Other notation is as before. The global structure of the algorithm is as follows.

```
Initially b(R') = 1, p(R') = Θ(n).
While b(R') ≤ n/λ(n)
  execute Step 1
While p(R') > 4n/λ(n)
  execute Step 2
Execute Step 3
```

Step 1: Restricted Spiral Search

Actually, this is not nearly as straightforward as it was for trees. We would like the robot to iteratively explore all parts of the environment within distance $1, 2, 4, \ldots$ until $b(R')$ gets too large. The problem is that the robot must be careful to keep track of the distance to each point it encounters; that is, it must know a *shortest* path back to the origin γ_0 for each point it reaches. Fortunately, there is a "compact search" subroutine due to Betke, Rivest, and Singh [BRiSi] which accomplishes just this.

Thus Step 1 will proceed as follows. For successive values of 2^j $(j = 0, 1, 2, \ldots)$ the robot explores all points within 2^j of $Z(\gamma_0)$. We implement stage j of this process using a simple modification of the compact search algorithm of [BRiSi] — the robot turns back whenever it is about to move more than 2^j from γ_0.

Lemma 4.12 *At the end of Step 1, the robot has traveled $O(b(R'))$ times the length of the optimal solution.*

Proof. The main result of [BRiSi] is that the robot will travel at most 10 times the total length of all edges in the region searched. In stage j, each rectangle (or partial rectangle) has perimeter at most $4 \cdot 2^j$ and there are at most $b(R')$ rectangles; thus the robot travels at most $40b(R') \cdot 2^j$. Summing over all stages, the distance traveled is at most $80r(R')b(R')$; meanwhile, the optimal solution has length at least $r(R')/2$. ∎

Step 2: Extending Multiple Critical Paths

Let $c(R')$ denote the quantity

$$\min_{(\alpha \in R')} \min_{(v,v' \in D_\alpha)} d_R(v, v').$$

That is, $c(R')$ is the smallest distance between two vertices in R corresponding to the same vertex in R'. Let P denote the polygonal path from v to v'. As in Section 4.2, the robot, starting from v, tries to follow as many copies of P in succession as possible. After this, $c(R')$ and the values of v, v' are updated, and the robot iterates. Step 2 comes to an end when $p(R')$ gets down to $\frac{4n}{\lambda(n)}$.

To bound the distance traveled in Step 2, we first prove a combinatorial lemma about trees with edge lengths. Let τ be a tree with m vertices and maximum degree Δ, and let ρ be a length function on its edges. If v is a vertex of τ, $B_d(v)$ will denote the set of all vertices within distance d of v and $rad(\tau)$ will denote, as usual, the smallest d such that there exists a v with $\tau \subset B_d(v)$.

Lemma 4.13 *There exists a vertex v^* of τ for which*

$$\left| B_{\frac{rad(\tau)}{\lambda(m)^2}}(v^*) \right| \geq \frac{\log m}{\Delta}.$$

Proof. Let u be a vertex of τ that realizes the radius; that is, $B_{rad(\tau)}(u) \supset \tau$. For $i = 1, \ldots, \lambda(m)^2$, let τ_i denote the set of all vertices in τ whose distance from u is at most $\frac{i}{\lambda(m)^2} \cdot$

$rad(\tau)$, and $m_i = |\tau_i|$. Setting $m_0 = 1$, we know that

$$\prod_{i=1}^{\lambda(m)^2} \frac{m_i}{m_{i-1}} = m$$

so by the pigeonhole principle there is some j for which $\frac{m_{j+1}}{m_j} \geq \log m$.

Consider cutting off the tree at distance $\frac{j}{\lambda(m)^2} \cdot rad(\tau)$ from u, and let $e_1, \ldots, e_s$ be the edges that cross this boundary. Since T has maximum degree Δ, and there are only m_j vertices in τ_j, we have $s \leq \Delta m_j$; now, since $m_{j+1} \geq m_j \log m$, one of the subtrees below some e_k has at least $\log m/\Delta$ vertices in $\tau_{j+1} - \tau_j$. Thus we can let the vertex in the subtree below e_k which is closest to u be v^* in the statement of the lemma. ∎

Lemma 4.14 *In Step 2, the robot travels at most $O(\frac{n}{\lambda(n)})$ times the length of the optimal solution.*

Proof. Consider building a shortest-paths tree, rooted at γ_0, on the vertices of the search region R'. (Note that we can build such a tree since we used a compact search algorithm in Step 1.) This tree has maximum degree four (since the obstacles are rectangles), so, applying Lemma 4.13, we see that there is some vertex α^* with at least $\frac{1}{4}(\log n - \log \lambda(n))$ vertices within a radius $\frac{r(R')}{\lambda(n)^2}$. Now suppose that there are not two vertices $v, v' \in D_{\alpha^*}$ for which

$$d_R(v, v') \leq \frac{2r(R')}{\lambda(n)^2}. \tag{4.1}$$

Then we could pack into R a collection of disjoint balls of radius $\frac{r(R')}{\lambda(n)^2}$, each of which contains at most one member of D_{α^*} and at least $\frac{1}{4}(\log n - \log \lambda(n))$ vertices of R. But this would imply

$$|D_{\alpha^*}| = p(R') \leq \frac{4n}{\log n - \log \lambda(n)} \leq \frac{4n}{\lambda(n)}$$

and thus the robot would not execute Step 2 at all.

Thus we have shown that as long as $b(R') \geq \frac{n}{\lambda(n)}$ and $p(R') \geq \frac{4n}{\lambda(n)}$, there will be vertices $v, v' \in D_{\alpha^*}$ satisfying Equation (4.1). As before, let P denote the polygonal path from v to v', and suppose that the robot traverses P^s beginning at $Z(\alpha^*)$ but is not able to traverse P^{s+1}. We will call this a *short iteration* if $s \leq \lambda(n)$, and a *long iteration* otherwise.

Observe that a short iteration eliminates at least one of v, v' from D_{α^*}, so there are at most n short iterations. Also, the length of P is at most $\frac{2r(R')}{\lambda(n)^2}$, so the robot travels at most $\frac{4r(R')}{\lambda(n)}$ in one such iteration, for a total of $\frac{4n}{\lambda(n)} \cdot r(R')$.

Now we claim that there can be at most one long iteration in Step 2. Indeed, at the end of such an iteration each $v \in D_{\alpha^*}$ P-covers at least $\lambda(n)$ vertices, so by arguments strictly analogous to those in the proof of Lemma 4.9, there can be at most $\frac{n}{\lambda(n)}$ vertices in D_{α^*}. Thus, Step 2 will come to an end after the first long iteration. Moreoever, s is always at most n, so the robot will travel at most $\frac{4n}{\lambda(n)^2} \cdot r(R')$ in this iteration.

As the length of the optimal solution is at least $r(R')/2$, the bound follows. ∎

Step 3 is implemented just as before. Thus, by Lemmas 4.12, 4.14, and the analogue of Lemma 4.10 for rectangles, we have

Theorem 4.15 *The above algorithm is $O(\frac{n}{\lambda(n)})$-competitive for the localization problem in an environment of n rectangles.*

4.4 Placing Unique Landmarks

Until now, we have been considering the "drop-off" version of the problem [TPBHSS], in which the robot is placed in an environment with very little starting information. But another situation in which localization arises is that of a robot which must repeatedly perform tasks in the same environment, and must begin by determining its current location. In such situations, it is useful to place k unique landmarks in the environment, so that the robot immediately knows where it is upon encountering one of them. It is not difficult to make these notions precise in our model. Let us simply say that a *k-marking* of the environment $\mathcal{E}$ is a function μ from the vertices of the environment to the set $\{0, 1, \ldots, k\}$; exactly one vertex gets each value $j = 1, \ldots, k$, and the rest get the value 0. Each time the robot gets to a new vertex v, it can determine the value $\mu(v)$.

For the sake of concreteness, let us consider the geometric trees of Section 4.2. The goal here is, for fixed k, to give a k-marking μ and an accompanying localization algorithm which achieves the lowest possible competitive ratio. Trying out this notion on the examples of Section 4.1, one finds that different environments can have strikingly different behaviors with respect to this measure. For example, one must place at least $\sqrt{n} - o(\sqrt{n})$ landmarks in the tree of Figure 4-2 before bringing about an asymptotic improvement in the best competitive ratio attainable (since there will have to be a landmark in all but a $o(1)$ fraction of the $\sqrt{n}$ "blocks"). On the other hand, by placing a *single* landmark at the exact center of the long path Π in Figure 4-3, one brings the best attainable competitive down from $\sqrt{n}$ to 9 (the robot simply uses two-way spiral search until it hits this landmark or one end of Π).

In light of this, we believe that the algorithmic question of finding the optimal k-marking, given a tree T or a set of rectangles R, is very interesting; we leave it as an open problem. In the remainder of this section, we turn to statements that can be made in general for environments of trees and rectangles.

Proposition 4.16 *For each tree T, there is a k-marking and a localization algorithm which is $O(\frac{n}{k})$-competitive.*

Proof. This is not difficult to prove directly; and using a lemma from [LoS], we can actually prove the stronger statement that there is, in effect, a single marking which works for all k. Specifically, it is proved in [LoS] that there is a numbering ψ of the n vertices of T (i.e. a bijection from V to $\{1, \ldots, n\}$) so that for each k, the removal of the vertices numbered 1 through k results in a forest in which no component has more than $\frac{2n}{k}$ vertices.

Given ψ, we define the k-marking μ_k in the natural way:

$$\mu_k(v) = \begin{cases} \psi(v) & \text{if } 1 \leq \psi(v) \leq k \\ 0 & \text{otherwise} \end{cases}$$

Given this marking, the localization algorithm is rather unsubtle: the robot performs spiral search until $p(T')$ decreases to 1 or it reaches a landmark (at which point $p(T')$ immediately equals 1). Since it is traveling in a component with at most $\frac{2n}{k}$ vertices, $b(T')$ will never exceed $\frac{2n}{k}$, and the result follows. ∎

Unfortunately, we see that no stronger statement can be made at this level of generality for trees, since the trade-off in Proposition 4.16 is tight up to constant factors for the tree of Figure 4-2 when $k = \sqrt{n}$. For the case of rectangular obstacles, we can prove a similar trade-off non-constructively.

Proposition 4.17 *For each environment R of n rectangles, there is a k-marking and a localization algorithm which is $O(\frac{n \log n}{k})$-competitive.*

Proof. Note that the statement is trivially true if k is not at least $\Omega(\log n)$, so we will assume that it is in what follows. Also, we only consider markings in which landmarks are placed at all four corners of $k/4$ rectangles. Thus, by abuse of notation, we will also speak of $\mu(R_i)$, where R_i is a rectangle. The localization algorithm will be the same as in Proposition 4.16: perform spiral search as in Step 1 of Section 4.3 until $p(R')$ decreases to 1 or a landmark is reached. To prove the stated bound, we must show that for some marking μ, the robot will travel no more than $O(\frac{n \log n}{k})$ times the length of the optimal solution, regardless of its starting position.

In fact, we claim that if μ is constructed by randomly marking $k/4$ rectangles, it will have this property with high probability. To prove this, we define a numbering ψ_v of the rectangles for each of the $4n$ vertices in the environment: ψ_v will tell the order in which the rectangles are encountered when the spiral search algorithm is performed beginning at v (i.e. $\psi_v(R_i) - 1$ rectangles are encountered before rectangle R_i, starting from v). Say that μ has Property Ψ if

$$\forall v \; \exists R_i : \mu(R_i) > 0 \text{ and } \psi_v(R_i) \leq \frac{8n \ln n}{k}.$$

The probability that μ fails to have Property Ψ for a single vertex v is at most

$$\left(1 - \frac{8 \ln n}{k}\right)^{\frac{k}{4}} \leq e^{-2 \ln n} = \frac{1}{n^2}$$

and so the probability that μ fails to have Property Ψ for any vertex is at most $\frac{4}{n}$.

So consider a marking μ which does have Property Ψ. Regardless of the robot's starting location, it will encounter a landmark by the time $b(R')$ reaches $\frac{8n \ln n}{k}$; the result now follows from Lemma 4.12. ∎

Appendix: Proofs of Claims 4.5 and 4.6

Recall that a subset $U = \{u_1, \ldots, u_k\}$ of the vertices of T induces a periodic path if $T(U)$ is a simple path and there is some polygonal path P such that $T(u_i, u_{i+1}) = P^{m_i}$ for natural numbers $m_1, \ldots, m_{k-1}$; and U induces a *comb tree* if $T(U)$ consists of a periodic path on vertices

$\{v_1, \ldots, v_k\}$ distinct from U together with disjoint paths $T(u_i, v_i)$, all of which are mutually congruent.

Proof of Claim 4.5. Let S denote $T(r, x, y, z)$, P denote the polygonal path $T(r, x) \simeq T(y, z)$, and Q denote $T(r, y) \simeq T(x, z)$. Let us suppose by induction that the claim holds for all examples in which the total number of edges in P and Q together is smaller than in S (note that the result clearly holds when P and Q each consists of a single edge). We consider three separate cases, based on the number of leaves in S.

Case 1: S has two leaves; suppose that these are r and z (other cases are similar). Then since C has no branch-vertices in this case, the polygonal paths PQ and QP are congruent. It is straightforward to show that this implies PQ is a periodic path.

Case 2: S has three leaves. This is actually impossible; suppose that x, y, and z are all leaves (other cases are symmetric). Then since x is a leaf, P^{-1} and Q have the same initial direction; since z is a leaf, P^{-1} and Q^{-1} have the same initial direction; and since y is a leaf, P and Q^{-1} have the same initial direction. Thus P and Q have the same initial direction, which implies that r must be a leaf.

Case 3: S has four leaves. Then P, Q, P^{-1}, and Q^{-1} all have the same initial direction. We eliminate the shortest of these four initial edges, and shorten the other three initial edges by the same amount. In this way, we have an example S' with one fewer edge in P or Q, so the claim holds for S'. This in turn implies that S is a comb tree. ∎

Proof of Claim 4.6. If all triples form periodic paths, then clearly $T(U)$ is a periodic path. So suppose $T(x, y, z)$ is a comb tree. Then since $T(x, y)$ has the same direction at both ends, $T(x, y, z')$ must be a comb tree for all $z' \in U$. Now if $T(x, p, q)$ is a periodic path, then $T(x, p)$ cannot have the same initial direction at both ends; thus $T(x, p, q')$ is a periodic path for all $q' \in U$. From this it follows that if any triple constitutes a comb tree, then all triples do. Also, we can conclude that all vertices are leaves in $T(U)$.

Next, observe that for all $z, z' \in U$, x has the same support point in $T(x, y, z)$ and $T(x, y, z')$ (it is the maximal sequence of line segments which is the same at the beginning and end of $T(x, y)$). Thus, it has the same support point in all comb trees. We now conclude that all the vertices in U have disjoint congruent paths joining them to the rest of $T(U)$. Deleting these paths, we obtain $|U|$ support points joined by periodic paths. Thus $T(U)$ is a comb tree. ∎

Chapter 5

Real-Time Server Algorithms

The trade-off between the computational resources of an on-line algorithm and the competitive ratio it can achieve is a question that has received relatively little attention. In some sense, it is not hard to see why this should be the case: proving lower bounds for traditional algorithms is notoriously difficult, and introducing the on-line/off-line distinction only seems to add to the complications. Thus establishing a non-trivial relationship between two measures such as running time and competitive ratio appears to be mainly beyond the reach of current techniques.

Nevertheless, it seems worth considering the problem at least for the simplest kind of on-line algorithms. What we have in mind is the following. Consider an on-line problem that can be expressed in the framework set up in Chapter 2: an algorithm $\mathcal{A}$ is presented with a sequence of requests $\sigma = \{\sigma_1, \sigma_2, \ldots\}$ in order; when σ_i is presented, $\mathcal{A}$ must perform some computation. We are interested in considering algorithms $\mathcal{A}$ that can meet some fixed time bound in processing each request; in particular, $\mathcal{A}$ should not slow down as more and more requests are presented. Thus, we (informally) define a *real-time* algorithm to be one which uses a constant amount of space, and constant time per request σ_i. Of course, the notion "constant" will depend on the particular problem, but the running time of $\mathcal{A}$ should in any case not depend on the number of requests seen so far.

The 2-server problem appears to be a good one on which to try out these notions. First of all, a fair amount is known about the bounds one can achieve — there are at least three optimal 2-server algorithms in the literature [MMS, CL1, CL4] — and a number of constant-time algorithms have been analyzed [IR, CL2, CKPV]. Also, the structure of the 2-server problem is in some sense quite a bit simpler than the structure of the general k-server problem.

In this setting, a real-time algorithm is one which decides which server to send to each request using an amount of time and space independent of the cardinality of the metric space and of the number of points requested so far. None of the known optimal 2-server algorithms have this property; for example, in an infinite metric space such as $\mathbf{R}^d$, they slow down with each additional request. Indeed, currently the best real-time algorithm known has a competitive ratio of 4 [CL2] (improving on a 10-competitive algorithm [IR]); and until now, the only known metric spaces in which a fast algorithm could achieve the optimal ratio of 2 were those which could be embedded in a tree.

All of this leads to a very interesting open question: is there a real-time 2-server algorithm with a competitive ratio of 2? We do not settle this question here; however, we give two results which provide evidence in opposite directions. Note also that the question is essentially independent of the k-server conjecture, which asks how well an on-line algorithm can perform regardless of the amount of computation it requires.

First we present a real-time, memoryless 2-competitive algorithm for two servers in $\mathbf{R}^d$, for any dimension d, under the L_1 ("Manhattan") metric. This result considerably extends the class of metric spaces for which there is known to be a fast, optimal algorithm. Moreover, the original real-time 2-competitive algorithm for trees was based on a very elegant memoryless strategy, "Double-Coverage," that was first discovered for servers on a line [CKPV] (this algorithm was discussed in Chapter 2). However, attempts to extend this technique to produce optimal deterministic algorithms in spaces more general than trees had so far not been successful. Thus, an interesting feature of our main result is that it follows the style of Double-Coverage, and can therefore be viewed as an extension of that algorithm to the case of two servers in higher dimensions.

In Section 5.2, we turn to the question of lower bounds. A number of papers have proposed "balancing" algorithms for the k-server problem. The basic balancing algorithm works as follows: for each server s_i, its total distance traveled is maintained in the variable D_i; when a request is made at a point r, the algorithm sends the server which minimizes $D_i + rs_i$ (for $x, y \in M$, let xy denote the distance between them). This rule was shown to be k-competitive for k servers when the cardinality of the request space M is $k+1$ [MMS], and for the "weighted-cache" problem, which includes the paging problem as a special case [CKPV].

However, the algorithm does not achieve any constant competitive ratio, even for two servers, in a general metric space M. Thus it was somewhat surprising that a rule minimizing the quantity $D_i + 2rs_i$ was shown to be 10-competitive for two servers [IR]. A later construction showed that this algorithm is no better than 6-competitive [CL2].

Here, we show a new lower bound for the class of balancing algorithms in general. Let $f : \mathbf{R}^+ \to \mathbf{R}$, and B_f be the server algorithm which does nothing when the request point is already covered, and otherwise moves the server which minimizes $D_i + f(rs_i)$, where D_i is the total distance traveled by server i. We will describe B_f as a balancing algorithm with cost function f. Observe that we make no restrictions whatsoever on the nature of the function f.

Our main result is a lower bound of $(5+\sqrt{7})/2$ (~ 3.82) on the competitive ratio of *any* such balancing algorithm for two servers. In view of the 2-competitive algorithms of [MMS, CL1, CL4], this shows that no optimal on-line 2-server algorithm can be expressed as a decision rule B_f for any f. To our knowledge, this represents one of the first lower bounds for on-line algorithms based solely on computational resources.

We use the following notation. For a server algorithm $\mathcal{A}$, let $\Gamma(\mathcal{A})$ denote its competitive ratio. If $\mathcal{A}$ is c-competitive for some c, we will say it is *competitive*; otherwise, we write $\Gamma(\mathcal{A}) = \infty$. As before, OPT will denote the optimal off-line server algorithm, whose cost on a request sequence σ is always the minimum possible; we sometimes refer to OPT as the "adversary" algorithm.

5.1 Two Servers in Euclidean Space

Let x^i denote the i^{th} coordinate of $x \in \mathbf{R}^d$. For two points $x, y \in \mathbf{R}^d$, define $\tau_d(x,y)$ to be the closed d-dimensional box (possibly degenerate) with x and y as opposite corners. That is, $w \in \mathbf{R}^d$ lies in $\tau_d(x,y)$ if and only if w^i is between $\min(x^i, y^i)$ and $\max(x^i, y^i)$ for $i = 1, \ldots, d$. A point $z \in \mathbf{R}^d$ induces the following partial order $\preceq_z$ on $\mathbf{R}^d$: $x \preceq_z y$ if and only if x is contained in the box $\tau_d(y,z)$. It is easily verified that $\preceq_z$ satisfies the properties of a partial order.

First consider the following problem. We have two sets P and Q, each consisting of n points in $(\mathbf{R}^d, L_1)$, and we wish to find a minimum-cost perfect matching between them. The following lemma is central to the analysis of the algorithm.

Lemma 5.1 *For each $q \in Q$, there is a minimum-cost matching M in which q is matched to one of the points of P which is minimal with respect to $\preceq_q$.*

Proof. Consider a matching M' in which $p' \in P$ is matched to q, and p' is not minimal with respect to $\preceq_q$. In this case, there is some minimal $p \in P$ such that $p \preceq_q p'$; suppose that p is matched to some $q' \in Q$. We can draw a shortest path from p' to q that passes through p; thus the matching M in which p is matched to q and p' is matched to q' has cost at most that of M'. ∎

The Algorithm in the Plane

We first give the algorithm in the plane, and then show how to extend it to higher dimensions. We number the quadrants of the plane in clockwise order, beginning with the positive quadrant. As is standard in analyzing server algorithms, we view the request sequence as a game between the algorithm and an omniscient adversary. In each round of the game, the adversary generates a request and serves it optimally (using knowledge of the future requests). The algorithm then must also serve the request, without knowing anything about the behavior of the adversary servers or the remainder of the request sequence.

Assume that the algorithm's servers are at points s_1 and s_2, the adversary servers are at points o_1 and o_2, and a request comes to some point in the plane. Since the L_1 metric is invariant under translations and 90-degree rotations, we can assume that the request is at the origin and that the on-line server closer to the request, s_1, lies in the first quadrant.

First, the algorithm computes the partial order $\preceq$, with respect to the origin, on the two servers. Clearly s_1 will be a minimal element. If s_2 is not also minimal, we simply move s_1 along a shortest path to the origin. Otherwise, both are minimal and we have the following cases:

1. s_2 lies in the third quadrant. Here we move both servers at constant speed along shortest paths to the origin until s_1 reaches it.

2. s_2 lies in the second quadrant. Here, we move both servers at constant speed to the positive y-axis — if s_2 reaches it first, we also move it down the y-axis — until s_2 is no longer minimal.

3. s_2 lies in the fourth quadrant. This is strictly analogous to the previous case.

4. s_2 also lies in the first quadrant. Then consider the rectangle $r_2(s_1, s_2)$, and let y be the corner of this rectangle closest to origin. We move both servers toward y until s_1 reaches it; now s_2 is no longer minimal.

Theorem 5.2 *The above algorithm is* 2*-competitive in the* L_1 *plane.*

Proof. We analyze the algorithm's performance using the CDRS potential function [CDRS]

$$\Phi = s_1 s_2 + 2M_{\min}(S, OPT),$$

where $M_{\min}(S, OPT)$ denotes the value of the minimum-cost matching between the adversary servers and the on-line servers, and $s_1 s_2$ is, as before, the distance between s_1 and s_2. By the standard potential-function argument, the following two facts will imply that the algorithm is 2-competitive.

1. When the adversary moves, paying c, it can raise Φ by at most $2c$.

2. When the algorithm moves, paying c', it lowers Φ by at least c'.

The first of these facts holds in any metric space, and has been proved in earlier papers [CKPV, CL3, CDRS]; the argument is simply that the value of the minimum-cost matching goes up by at most c' while the first term of Φ is untouched.

Now consider the second fact. We break the behavior of the algorithm into two *phases.* In the first (possibly empty) phase, both servers are moving; in the second phase, only one server is moving. In the second phase, in which one server moves a distance d, the value of first term of Φ increases by at most d, while Lemma 5.1 implies that the value of the minimum-cost matching goes down by d. Thus Φ goes down by at least d, as desired.

When the algorithm is moving both servers (each a distance d') the matching component cannot go up (whichever server is matched to the origin is decreasing its contribution to the cost optimally). Meanwhile, examining the four cases above, we see that the first term of Φ goes down by $2d'$. Thus, Φ decreases by at least $2d'$, and the proof is complete. ∎

Under the Euclidean metric, the on-line servers can follow the above algorithm and pay at most the L_1 cost. Meanwhile, the adversary only has to pay the L_2 cost, which can be smaller by at most a factor of $\sqrt{2}$. Thus we have

Corollary 5.3 *The algorithm is $2\sqrt{2}$-competitive (~ 2.83) in the Euclidean plane.*

The Algorithm in d Dimensions

Consider now the case of two servers in $\mathbf{R}^d$, under the L_1 metric. As before, by a translation of $\mathbf{R}^d$, we may assume that the request is at the origin; we also relabel the servers if necessary so that s_1 is closer to the request.

The algorithm in d dimensions works as follows. If only one server is minimal, it moves on a shortest path to the origin while the other remains fixed. Otherwise, both servers are minimal. Let $s_1 = (s_1^1, \ldots, s_1^d)$ and $s_2 = (s_2^1, \ldots, s_2^d)$ in coordinates; we proceed in the following two stages:

1. As long there is a j for which s_1^j and s_2^j have opposite sign, we move the servers at the same speed towards the hyperplane $\{x : x^j = 0\}$ so as to bring one of these values to 0.

2. The servers begin the second stage with s_1^j and s_2^j having the same sign for each j; that is, they lie in the same closed orthant. By relabeling the axes, we may assume this is the positive orthant. Let y be the corner of $\tau_d(s_1, s_2)$ closest to the origin; that is,

$$y = (\min(s_1^1, s_2^1), \ldots, \min(s_1^d, s_2^d)).$$

   As before, denote the j^{th} coordinate of y by y^j. The servers repeatedly execute the following step: each picks a j for which $s_i^j - y^j$ is strictly positive, and moves at constant speed towards the hyperplane $\{x : x^j = y^j\}$ so as to decrease s_i^j to y^j. Each such move reduces the number of coordinates j for which $s_i^j - y^j > 0$ for one of the servers s_i. Thus, after at most d such steps, server s_1 will be sitting on y. At this point, only s_1 is minimal, and we proceed as above.

Theorem 5.4 *The algorithm is 2-competitive under the L_1 metric in $\mathbf{R}^d$.*

Proof. We use the same potential function Φ. The analysis of the adversary's move and the case in which only one on-line server is minimal are handled as in the proof for two dimensions. When both on-line servers move a distance d, we must show that the potential Φ drops by at least $2d$. This will be implied by the following property:

($*$) Both servers are moved a distance d closer to the origin and $2d$ closer to each other simultaneously.

The motion of the servers in the first stage of the algorithm clearly satisfies this property. Observe that throughout the second stage, there is a shortest path from s_1 to s_2 that passes through the corner y of $\tau_d(s_1, s_2)$. But each step in this stage brings a server simultaneously closer to the origin and to y; thus Property ($*$) is maintained in the second stage as well. ∎

Corollary 5.5 *The algorithm is $2\sqrt{d}$-competitive under the Euclidean metric in $\mathbf{R}^d$.*

5.2 A Lower Bound for Balancing Algorithms

To prove the lower bound, we begin by "classifying" the function f used in the balancing rule in a certain way. We will say that a property P of positive real numbers holds "e.f." (everywhere but in a finite interval) if $\exists x_0 \forall (x \geq x_0) P(x)$. Similarly P holds "a.l." (for arbitrarily large reals) if $\forall x_0 \exists (x \geq x_0) P(x)$. For any function $f : \mathbf{R}^+ \to \mathbf{R}$, and $p \in \mathbf{R}^+$, say that $f \in \varphi(p)$ if $f(x) \leq px$ e.f. If for some such p, $f \in \varphi(p)$, we will write $\rho_f = \inf\{p : f \in \varphi(p)\}$; $\rho_f = \infty$ otherwise. Observe that f does not necessarily belong to $\varphi(\rho_f)$, since the infimum is not necessarily attained.

Finally, the behavior of B_f is ambiguous when $D_i + f(rs_i) = D_j + f(rs_j)$ for $i \neq j$, and i, j both minimize this expression. The standard convention here is to let the adversary break the tie; in any event, none of our constructions rely on such degeneracies.

For the remainder of this section, all server algorithms will be 2-server algorithms.

Lemma 5.6 *If B_f is competitive, then $f(x) > 0$ e.f.*

Proof. Assume by way of contradiction that $f(x) \leq 0$ a.l. and that B_f is c-competitive for some c. Choose x_0 so that $f(x_0) \leq 0$ and N_0 large enough so that $N_0 \geq cx_0$ and $N_0 > -f(x_0)$. Since $f(x) \leq 0$ a.l., we can find an $N \geq N_0$ such that $f(N) \leq 0$.

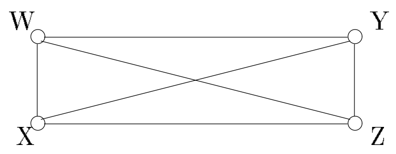


Figure 5-1: Metric Space M_1

Consider the metric space M_1 in Figure 5-1, with $WX = YZ = x_0$, $WY = WZ = XY = XZ = N$. Server s_1 begins on W and server s_2 on X. The first request is to the point Y. Since both servers are the same distance from Y, we may assume that s_1 responds. Now we request point Z; since

$$N + f(x_0) > 0 \geq f(N),$$

server s_2 will respond. At this point, B_f has paid $2N \geq 2cx_0$, while OPT, by using one server to cover requests to W, X and the other to cover Y, Z, need pay only x_0. Moreover, the situation is now symmetric to the beginning, so we can repeat the process indefinitely, contradicting the claim the B_f is c-competitive. ∎

Armed with this lemma, we can prove a much stronger result, generalizing the observation that the basic balancing algorithm is not competitive.

Lemma 5.7 *If B_f is competitive, then $\rho_f > 1$.*

Proof. Assume by way of contradiction that $\rho_f \leq 1$ and B_f is c-competitive. We have two cases to consider: $f \in \varphi(1)$ and $f \notin \varphi(1)$.

If $f \in \varphi(1)$, then for some $x_0 > 0$, $0 < f(x) \leq x$ for all $x \geq x_0$. Consider metric space M_1, with $WX = YZ = x_0$, all other lengths equal to $N = cx_0$, and the servers initially on W, X. Suppose we request point Y and s_1 responds. If we now request point Z, then since

$$N + f(x_0) > N \geq f(N),$$

server s_2 will respond. Moreover, this process can be continued indefinitely. Thus B_f pays $2cx_0$ in each round of this sequence, while OPT need pay only x_0, as in the above arguments.

If $f \notin \varphi(1)$, then we can still find some $y > 0$ such that $0 < f(x) < (1 + \frac{1}{2c})x$ for all $x \geq y$.

Moreover, since $f \notin \varphi(1)$, we can find some $x_0 > y$ such that $f(x_0) > x_0/2$. Construct metric space M_1 with $WX = YZ = x_0$, all other lengths equal to cx_0, put s_1 on W, s_2 on X, and request point Y. Suppose s_1 responds; we now request point Z. Since $f(x_0) > x_0/2$,

$$cx_0 + f(x_0) > (c + \frac{1}{2})x_0 = (1 + \frac{1}{2c})cx_0 > f(cx_0),$$

the last inequality following since $cx_0 \geq y$. Thus s_2 will respond. In this way, B_f pays $2cx_0$ on this sequence, while OPT need only pay x_0. ∎

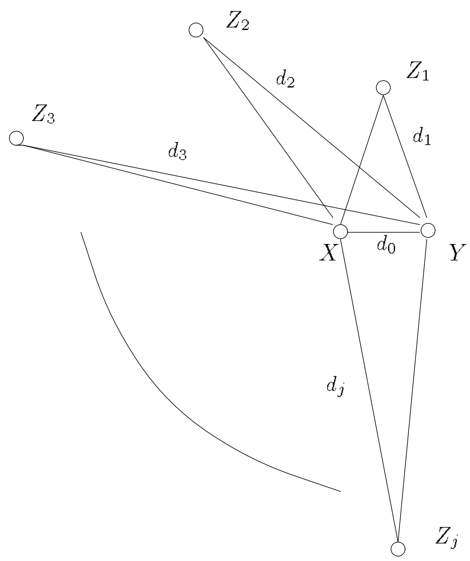


Figure 5-2: Metric Space M_2

Lemma 5.8 *If $\rho_f = p > 1$, then $\Gamma(B_f) \geq p/2 + 3/2$.*

Proof. We show that for each $\beta > 0$, there exist arbitrarily long request sequences σ for which $B_f(\sigma) > (p/2 + 3/2 - \beta)OPT(\sigma)$; this will establish the lemma. We construct a countably infinite metric space M_2 as follows. The "core" of the space is a short segment XY of length d_0, chosen so that $pd_0/2 < f(d_0) < 2pd_0$. The sequence σ is built in phases; in phase j, $j = 1, 2, \ldots$, all the requests are to X, Y, and a point Z_j at distance d_j from both X and Y. (See Figure 5-2.) Let σ_j denote the sequence up through the end of phase j.

Let D_j denote $\sum_{i=0}^{j-1} d_i$, and D'_j denote the sum of D_j and the total distance traveled by both on-line servers in all previous phases. We fix some very small $\alpha > 0$, to be determined below, and since $\rho_f = p$, we can choose d_j such that $D'_j < \alpha d_j$, and $f(d_j) > (p - \beta/2)d_j$. Phase j begins with the two on-line servers sitting on X and Y. First the point Z_j is requested; then the points X and Y are requested repeatedly until the on-line server at Z_j returns to the core XY; at this point the phase comes to an end.

In this phase, one adversary server moves out to Z_j and immediately returns on the next request, for a cost of $2d_j$. Meanwhile, the on-line server covering X and Y has built up a distance of no more than D'_j from all previous phases, so it must move more than

$$\begin{aligned} d_j + f(d_j) - (D'_j + f(d_0)) &> d_j + (p - \beta/2)d_j - (\alpha d_j + 2p\alpha d_j) \\ &= (p+1)d_j - (2p+1)\alpha d_j - (\beta/2)d_j \end{aligned}$$

before the server at Z_j is selected to return. Thus, the on-line servers move a distance of at least

$$(p+3)d_j - (2p+1)\alpha d_j - (\beta/2)d_j$$

in this phase, while the adversary servers move no more than $2d_j + 2\alpha d_j$ in *all* phases up to this one. By choosing α small enough, we can ensure that

$$\frac{B_f(\sigma_j)}{A(\sigma_j)} \geq \frac{(p+3) - (2p+1)\alpha - (\beta/2)}{2 + 2\alpha} \geq p/2 + 3/2 - \beta.$$

Since we can continue this construction for an arbitrary number of phases, the result follows.

∎

It is clear that a very similar construction involving metric space M_2 shows that if $\rho_f = \infty$, then B_f is not competitive. The final lemma is based on a construction in [CL2].

Lemma 5.9 *If $\rho_f = p > 1$, then $\Gamma(B_f) \geq 3p/(p-1)$.*

Proof. In the style of the previous proof, we show that for each $\beta > 0$,

$$\Gamma(B_f) > \frac{3p}{(p-1)} - \beta.$$

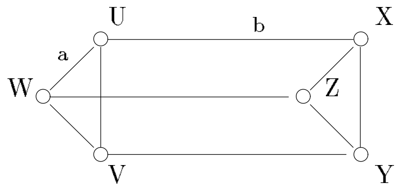


Figure 5-3: Metric Space M_3

Choose positive $\delta < \beta/2$ and

$$\varepsilon < \frac{2(p-1)^2}{4p-1} \cdot \delta.$$

Now take N large enough so that $f(x) < (p+\varepsilon)x$ for all $x > N$ and, since $\rho_f > p - \varepsilon$, we can find $a > N$ such that $f(a) > (p-\varepsilon)a$.

Consider the metric space M_3 of Figure 5-3. Triangles UVW and XYZ have sides of length a; the three edges crossing between the triangles have length

$$b = \left(\frac{2p+1}{2p-2} - \delta\right) a.$$

Put server s_1 on U and s_2 on V. We place the adversary servers on V and Z. Also, set $D_1 = a/2$ and $D_2 = 0$ (this can be easily accomplished by starting one of the on-line servers from a seventh point in the metric space).

We construct a request sequence as follows. First we request Z, which will be served by s_2. We then request X — as shown below, this will be served by s_1. At this point, D_2 exceeds D_1 by $a/2$, and the situation is symmetric to the beginning. The optimal algorithm can use one server to cover each triangle and hence pays a; B_f pays $2b + a$, for a ratio of

$$\frac{2b+a}{a} = 1 + \frac{2p+1}{p-1} - 2\delta > \frac{3p}{p-1} - \beta.$$

Thus we need only verify that s_1 will serve the request at X. This follows because

$$D_1 + f(b) = \frac{a}{2} + f\left(\left(\frac{2p+1}{2p-2} - \delta\right) a\right) \leq a\left(\frac{2p^2+2p-1}{2p-2} - p\delta + \frac{2p+1}{2p-2} \cdot \varepsilon\right),$$

which is less than or equal to

$$D_2 + f(a) = a + b + f(a) \geq a\left(\frac{2p^2 + 2p - 1}{2p - 2} - \delta - \varepsilon\right).$$

■

Theorem 5.10 *For all f, $\Gamma(B_f) \geq (5+\sqrt{7})/2$.*

Proof. If $\rho_f \leq 1$ or $\rho_f = \infty$, then B_f is not competitive. In the case where $1 < \rho_f < \infty$, Lemmas 5.8 and 5.9 imply that

$$\Gamma(B_f) \geq \max\{\frac{3\rho_f}{\rho_f - 1}, \frac{\rho_f + 3}{2}\}.$$

This expression is minimized at $\rho_f = 2 + \sqrt{7}$, with a value of $(5+\sqrt{7})/2$. ■

As noted above, we have the following interesting corollary.

Corollary 5.11 *No optimal on-line 2-server algorithm can be expressed as B_f for any function f.*

Chapter 6

Conclusion and Open Problems

6.1 Robot Navigation

We have considered two basic on-line navigation problems: searching for a goal in an unknown polygon, and determining one's location in an environment for which the map is known (*localization*). A number of questions are left open by this work; we summarize some of these below, along with a number of other possible directions for future work.

1. To begin with, there are gaps between the upper and lower bounds that we can show for the localization problem in both kinds of environments considered in Chapter 4. Certainly for the case of rectangular obstacles, it seems that there should be room for improvement on the upper bound. Indeed, it is quite possible that essentially the same algorithm given here actually achieves a much better competitive ratio; it appears that a stronger analysis of the geometry of the rectangle packing is needed.

   Also, as suggested in Section 4.4, the problem of placing landmarks brings up some interesting algorithmic questions. Here is one concrete decision problem: given a tree T and an integer k, does there exist a k-marking μ and an accompanying localization algorithm with competitive ratio at most 9? (9 appears to be the right constant to use here, as it is the best competitive ratio attainable for the search problem on a line.)

2. The results of Chapter 3 apply only to the case of *simple* rectilinear polygons; i.e. those with no holes. [DKP] conjectures that a constant competitive ratio should be attainable

for the problem of exploring a rectilinear polygon containing an arbitrary number of rectilinear holes. (They give an algorithm which is $O(k)$-competitive, where k is the number of holes.)

Since a number of variations on this problem are possible, let us make precise what we mean: we want the robot (whether on-line or off-line) to traverse a path from which it can *see* all points in free space *and* on the boundaries of the obstacles. This question appears to be rather difficult (and the off-line version is NP-Complete, by a simple reduction from the Traveling Salesman problem); perhaps a good starting point would be the special case of a large square containing an arbitrary number of unit squares as obstacles.

Of course, one could also consider the search problem in this setting; it is not clear in this case what kind of bounds one would be trying to achieve.

3. The problem of using multiple interacting robots effectively is one that comes up frequently in the larger robotics literature. Most of the previous theoretical work on multiple robots is somewhat orthogonal to these concerns; it has tended to deal with computationally limited robots that use one another mainly for "pebbling" purposes (see e.g. [BK] and the references therein).

   We are interested in the following sort of problem. One has k independent robots trying to solve a search problem in some geometric environment; the quantity being minimized in this situation is the time taken to find the goal, rather than the total distance traveled.[1] If the best competitive ratio attainable by a single robot in this environment is c, can one give algorithms which are $O(\frac{c}{f(k)})$-competitive, where $f(k)$ is a reasonably fast-growing function?

   This question resembles some of the load-balancing problems that have been considered from the perspective of competitive analysis; essentially we are asking how "parallelizable" the search problem is in the on-line setting.

4. Much of the appeal of Klein's street problem (Chapter 3) is in the definition of a street itself: it is a natural type of polygon in which a constant competitive ratio can be achieved

[1] For a single robot, time and distance tend of course to be interchangeable in most of these problems.

for the search problem. We are interested in looking for other types of polygons in which competitive search is possible.

One obvious possibility is the class of *star-shaped polygons* (P is star-shaped if there is some point inside P that can see the entire boundary): can a robot starting at s find a point t while traveling a constant factor times $d(s, t)$? A related and probably easier question is that of whether a constant competitive ratio can be achieved for the problem of finding a member of the *kernel* of P (i.e. the set of "star points" of P).

Of course, star-shaped polygons are a rather simple class, but studying them may suggest more general algorithms. For example, suppose we define the *illumination number* of P to be the minimum number of point-source "light bulbs" needed to provide direct illumination to the entire boundary of P from its interior (so P is star-shaped iff its illumination number is one); now we can look for algorithms that achieve a constant ratio for searching polygons with fixed illumination number.

5. What does it take to build a maze? Specifically, it is interesting to ask oneself what constitutes the difference between the "shortest-path" problems of [PY, BRaSc, BBFY], and the geometric search problems [K, Kl] of the type discussed in Chapter 3. Initially, one is tempted to conclude that the difference is in whether the coordinates of the goal are known; but, as any good labyrinth-designer knows, it is possible for the entrance and exit of a maze to be physically right next to each other, and for the maze still to be very difficult.

   In other words, when the obstacles can be sufficiently complex, there is no value in knowing the coordinates of the goal. We can make this more precise as follows. Let $\mathcal{F}$ denote a family of rectilinear polygons, which will be used as obstacles in the sense of [PY] (e.g. $\mathcal{F}$ could be the set of all rectangles, or the set of all L-shaped polygons, and so on). As usual, we will require that all members of $\mathcal{F}$ have at least unit thickness, and that they cannot be "glued together." Consider now the navigation problem in which the plane is filled with members of $\mathcal{F}$, and a robot starting at s is given the coordinates of a point t which it must travel to. Let n denote the straight-line distance from s to t; recall that this could be much less than $d(s, t)$, the length of the shortest obstacle-avoiding path.

Say that $\mathcal{F}$ is *maze-inducing* if for no function f is there an $f(n)$-competitive on-line algorithm; that is, the competitive ratio for the navigation problem cannot be bounded in terms of the straight-line distance from s to t. Thus, the results of [BRaSc] show that the class of rectangles is not maze-inducing. On the other hand, it is not difficult to build mazes when $\mathcal{F}$ is the set of all rectilinear polygons (with at least unit thickness). So a natural question would be to try giving some characterization of classes $\mathcal{F}$ which are maze-inducing. For example, is it true that no $\mathcal{F}$ consisting of polygons with a uniformly bounded number of vertices, or even a uniformly bounded illumination number, can be maze-inducing? Along these lines one could define, by analogy with the illumination number, the *segment number* of P to be the minimum number of line-segment light sources needed to provide direct illumination to the entire boundary of P from its interior. We can show it is possible to build mazes when $\mathcal{F}$ is the set of polygons with segment number at most three, but do not know whether it is possible when the segment number is at most two.

Of course, there are many other issues one could consider. As mentioned at the beginning of Chapter 4, most on-line navigation problems studied up to this point have maintained a fairly binary distinction between having a map and having no map. But many naturally arising robotics problems involve navigation when the robot has some limited information about its relation to the environment. The localization problem is a step in this direction, as are the papers of Donald [Don] and Blum and Chalasani [BC].

As in the case of several other on-line problems, there are numerous "design choices" one faces when formulating a navigation problem of the variety studied here. We believe it is important in this process to keep in mind the problems faced by designers of autonomous mobile robots. It is not necessary that the theoretical algorithms produced in this setting be directly implementable in contemporary robots; but they seem to be most valuable when they provide some insight into the structure of navigation problems faced in real-world applications.

6.2 Servers

We have addressed the notion of *real-time* 2-server algorithms — those that only use only a constant amount of space, and constant time per request. It is well-known that the best competitive ratio attainable by a 2-server algorithm is 2, and this can be achieved by several algorithms, all of which perform a substantial amount of computation on each request. We have shown a 2-competitive real-time algorithm in n dimensions, for every n, under the Manhattan metric. On the other hand, we have shown that the class of *balancing algorithms* (a subset of the real-time algorithms) cannot achieve a competitive ratio better than $\frac{5+\sqrt{7}}{2}$ in general.

There are some immediate questions one could consider in light of these results, as well as a more global one. Specifically, can our algorithm for the Manhattan metric be adapted to the standard Euclidean metric while maintaining the competitive ratio of 2? Can it be extended to the case of three or more servers? (We can show that most natural generalizations in this direction are not 3-competitive.) It would also be interesting to try improving the upper bounds for 2-server balancing algorithms; the lower bound we show in Chapter 5 is still below the best known competitive ratio for a general real-time algorithm.

But at a higher level, we believe the fundamental problem here is to decide whether or not there is a real-time 2-server algorithm with a competitive ratio of 2. We do not venture a conjecture; the results of the previous chapter present evidence for both possible answers. Of course, one could ask the analogous question for more than two servers, where all these problems become much murkier ... but the main point is this: lower bounds for on-line algorithms have typically been based simply on information-theoretic arguments alone; that is, regardless of how the algorithm makes its decisions, there will be input sequences on which it performs badly. The hope is that as we begin to understand this area better, we can also begin making distinctions based on the computational resources an algorithm has at its disposal.

Bibliography

[AA] N. Alon, Y. Azar, "On-line Steiner trees in the Euclidean plane," *Proc. 8th ACM Symposium on Computational Geometry*, 1992, pp. 337–343.

[AAFPW] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts, "On-line machine scheduling with applications to load balancing and virtual circuit routing," *Proc. 25th ACM Symposium on Theory of Computing*, 1993, pp. 623–631.

[AAP] B. Awerbuch, Y. Azar, S. Plotkin, "Throughput-competitive on-line routing," *Proc. 34th IEEE Symposium on Foundations of Computer Science*, 1993.

[AAPW] B. Awerbuch, Y. Azar, S. Plotkin, O. Waarts, "Competitive routing of virtual circuits with unknown duration," *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms*, 1994.

[ABFR] B. Awerbuch, Y. Bartal, A. Fiat, A. Rosen, "Competitive non-preemptive call control," *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms*, 1994.

[BCR] R. Baeza-Yates, J. Culberson, G. Rawlins, "Searching in the plane," *Information and Computation*, 106(1993), pp. 234–252.

[BBFY] E. Bar-Eli, P. Berman, A. Fiat, P. Yan, "On-line navigation in a room," *Proc. 3rd ACM-SIAM Symposium on Discrete Algorithms*, 1992, pp. 237–249.

[Be] R. Bellman, "A minimization problem," *Bulletin of the AMS*, 62(1956).

[BBKTW] S. Ben-David, A. Borodin, R. Karp, G. Tardos, A. Wigderson, "On the power of randomization in on-line algorithms," *Proc. 22nd ACM Symposium on Theory of Computing*, 1990, pp. 379–386.

[BRiSi] M. Betke, R. Rivest, M. Singh, "Piecemeal learning of an unknown environment," *Proc. Sixth ACM Conference on Learning Theory*, 1993, pp. 277–286.

[BC] A. Blum, P. Chalasani, "An on-line algorithm for improving performance in navigation," *Proc. 34th IEEE Symposium on Theory of Computing*, 1993.

[BCCPRS] A. Blum, P. Chalasani, D. Coppersmith, W. Pulleyblank, P. Raghavan, M. Sudan, "The minimum latency problem," *Proc. 26th ACM Symposium on Theory of Computing*, 1994, pp. 163–171.

[BRaSc] A. Blum, P. Raghavan, B. Schieber, "Navigating in unfamiliar geometric terrain," *Proc. 23rd ACM Symposium on Theory of Computing*, 1991, pp. 494–504.

[BK] M. Blum, D. Kozen, "On the power of the compass (or, Why mazes are easier to search than graphs)," *Proc. 19th IEEE Symposium on Foundations of Computer Science*, 1978, pp. 132–142.

[BIRS] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, "Competitive paging with locality of reference", *Proc. 23rd ACM Symposium on Theory of Computing*, 1991, pp. 249–259.

[BLS] A. Borodin, N. Linial, M. Saks, "An optimal on-line algorithm for metrical task systems," *Journal of the ACM*, 39(1992), pp. 745–763.

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

[CKPV] M. Chrobak, H. Karloff, T. Payne, S. Vishwanathan, "New results on server problems," *SIAM J. Discrete Math.*, 4(1991), pp. 172–181.

[CL1] M. Chrobak, L. Larmore, "A new approach to the server problem," *SIAM J. Discrete Math.*, 4(1991), pp. 323–328.

[CL2] M. Chrobak, L. Larmore, "On fast algorithms for two servers," *J. Algorithms*, 12(1991), pp. 607–614.

[CL3] M. Chrobak, L. Larmore, "An optimal on-line algorithm for k-servers on trees," *SIAM J. Computing*, 20(1991), pp. 144–148.

[CL4] M. Chrobak, L. Larmore, "The server problem and on-line games," in *On-Line Algorithms*, D. Sleator and L. McGeoch, Eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science (vol. 7), 1992, pp. 11–64.

[CDRS] D. Coppersmith, P. Doyle, P. Raghavan, M. Snir, "Random walks on weighted graphs, with applications to on-line algorithms," *Journal of the ACM*, 40(1993), pp. 421–453.

[Cox] I. Cox, "Blanche – an experiment in guidance and navigation of an autonomous robot vehicle," *IEEE Trans. Robotics and Automation*, 7(1991), pp. 193–204.

[DKP] X. Deng, T. Kameda, C. Papadimitriou, "How to learn an unknown environment I: the rectilinear case," Technical Report CS-93-04, Department of Computer Science, York University. (Preliminary version in *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, 1991, pp. 298–303.)

[Don] B. Donald, "On information invariants in robotics," Technical Report 93-1341, Department of Computer Science, Cornell University, 1993.

[Dr] M. Drumheller, "Mobile robot localization using sonar," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(1987), pp. 325–331.

[DRW] G. Dudek, K. Romanik, S. Whitesides, "Localizing a robot with minimum travel," *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms.*

[FFKRRV] A. Fiat, D. Foster, H. Karloff, Y. Rabani, Y. Ravid, S. Vishwanathan, "Competitive algorithms for layered graph traversal," *Proc. 31st IEEE Symposium on Foundations of Computer Science*, 1991, pp. 288–197.

[FRR] A. Fiat, Y. Rabani, Y. Ravid, "Competitive k-server algorithms," *Proc. 30th IEEE Symposium on Foundations of Computer Science*, 1990.

[GGU] M. Garey, R. Graham, J. Ullman, "Worst-case analysis of memory allocation algorithms," *Proc. 4th ACM Symposium on Theory of Computing*, 1972.

[Gr] E. Grove, "The harmonic on-line k-server algorithm is competitive," *Proc. 23rd ACM Symposium on Theory of Computing*, 1991, pp. 260–266.

[GMR] L. Guibas, R. Motwani, P. Raghavan, "The robot localization problem in two dimensions" *Proceedings 3rd ACM-SIAM Symposium on Discrete Algorithms*, 1992, pp 259–268.

[HS] M. Halldorsson, M. Szegedy, "Lower bounds for on-line graph coloring," *Proc. 3rd ACM-SIAM Symposium on Discrete Algorithms*, 1992, pp. 211–216.

[IK] C. Icking, R. Klein, "The two guards problem," *Proc. 7th ACM Symposium on Computational Geometry*, 1991.

[IW] M. Imase, B. Waxman, "Dynamic Steiner tree problem," *SIAM J. Discrete Math*, 4(1991), pp. 369–384.

[Ir] S. Irani, "Coloring inductive graphs on-line," *Proc. 31st IEEE Symposium on Foundations of Computer Science*, 1990, pp. 470–479.

[IR] S. Irani, R. Rubinfeld, "A competitive 2-server algorithm," *Information Processing Letters*, 39(1991), pp. 85–91.

[Is] J. Isbell, "An optimal search pattern," *Naval Research Logistics Quarterly*, 4(1957), pp. 357–359.

[J] D. Johnson, "Fast algorithms for bin packing," *Journal of Computer and System Sciences*, 8(1974), pp. 272–314.

[KMSY] M. Kao, Y. Ma, M. Sipser, Y. Yin, "Optimal constructions of hybrid algorithms," *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms*, 1994.

[KRT] M. Kao, J. Reif, S. Tate, "Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem," *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms*, 1993, pp. 441–447.

[KMRS] A.R. Karlin, M.M. Manasse, L. Rudolph, and D.D. Sleator, "Competitive snoopy caching" *Algorithmica*, 3(1988), pp. 70–119.

[KPR] A.R. Karlin, S.J. Phillips, P. Raghavan, "Markov paging", *Proc. 33rd IEEE Symposium on Foundations of Computer Science*, 1992.

[Kier] H. Kierstead, "The linearity of First-Fit for coloring interval graphs," *SIAM J. Discrete Math*, 1(1988), pp. 526–530.

[KT] H. Kierstead, W. Trotter, "On-line graph coloring," in *On-Line Algorithms*, D. Sleator and L. McGeoch, Eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science (vol. 7), 1992, pp. 85–92.

[K] R. Klein, "Walking an unknown street with bounded detour," *Computational Geometry: Theory and Applications*, 1(1992), pp. 325–351. (Preliminary version in *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, 1991, pp. 304–313.)

[Kl] J. Kleinberg, "On-line search in a simple polygon," *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms*, 1994.

[KP] E. Koutsoupias, C. Papadimitriou, "On the k-server conjecture," *Proc. 26th ACM Symposium on Theory of Computing*, 1994, pp. 507–511.

[LMH] M. Levine, I. Marchon, G. Hanley, "The placement and misplacement of you-are-here maps," *Environment and Behavior*, 16(1984), pp. 139–157.

[LT] R. Lipton, A. Tomkins, "On-line interval scheduling," *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms*, 1994.

[LoS] L. Lovász, M. Saks, "Lattices, Möbius functions, and communication complexity," *Proc. 29th IEEE Symposium on Foundations of Computer Science*, 1989, pp. 81–90.

[LST] L. Lovász, M. Saks, W. Trotter, "An on-line graph coloring algorithm with sublinear performance ratio," *Discrete Math*, 75(1989), pp. 319–325.

[LuS] V. Lumelsky, A. Stepanov, "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, 2(1987) pp. 403–430.

[MMS] M. Manasse, L. McGeoch, D. Sleator, "Competitive algorithms for server problems," *J. Algorithms*, 11(1990), pp.208–230.

[MAWM] D. Miller, D. Atkinson, B. Wilcox, A. Mishkin, "Autonomous navigation and control of a Mars rover," in *Automatic Control in Aerospace*, T. Nishimura, Ed., Oxford: Pergamon Press, 1989, pp. 111–114.

[Ore] O. Ore *Theory of Graphs*, Providence: AMS, 1962.

[PY] C. Papadimitriou, M. Yannakakis, "Shortest paths without a map," *Theoretical Computer Science*, 84(1991), pp. 127–150.

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

[ST] D. Sleator, R. Tarjan, "Amortized efficiency of list update and paging rules," *Comm. ACM*, 23(1985), pp. 202–208.

[TPBHSS] W. Thompson, H. Pick, B. Bennett, M. Heinrichs, S. Savitt, K. Smith, "Map-based localization: the 'drop-off' problem," *Proc. DARPA Image Understanding Workshop*, 1990, pp. 706–719.

[TG] C. Thorpe, J. Gowdy, "Annotated maps for autonomous land vehicles," *Proc. DARPA Image Understanding Workshop*, 1990, pp. 765–771.

[TWSY] J. Turek, J. Wolf, U. Schwiegelshohn, P. Yu, "Scheduling parallel tasks to minimize average response time," *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms*, 1994.

[Vi] S. Vishwanathan, "Randomized online graph coloring," *Proc. 31st IEEE Symposium on Foundations of Computer Science*, 1990, pp. 464–469.

[Wang] C. Ming Wang, "Location estimation and uncertainty analysis for mobile robots," *Proc. IEEE International Conference on Robotics and Automation*, 1988, pp. 1230–1235.

[YD] Y. Yacoob, L. Davis, "Computational ground and airborne localization over rough terrain," Univ. of Maryland Tech Report CS-TR-2788.

[Yao] A. Yao, "New algorithms for bin packing," *Journal of the ACM*, 27(1980), pp. 207–227.