The Localization Problem for Mobile Robots

Jon M. Kleinberg
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge MA 02139, USA

Abstract

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. Guibas, Motwani, and Raghavan have given geometric algorithms for the problem of enumerating locations for a robot consistent with a given view of the environment. Here, we provide an on-line algorithm for a robot to move within its environment so as to uniquely determine its location. The algorithm improves asymptotically on strategies based purely on the "spiral search" technique of Baeza-Yates, Culberson, and Rawlins; an interesting feature of our approach 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.*

1 Introduction

A fundamental task for an autonomous mobile robot is that of *localization* — determining its location in a known environment [7, 23, 25]. 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. The localization problem has been considered in a 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 surroundings. Here, localization is used to determine the starting location at the beginning of the task, and to maintain positioning information over time [7, 9, 25]. A similar use of localization is in analyzing aerial photographs to determine the location from which they were taken [26]. Localization is also used in the design of autonomous exploration vehicles, such as the current prototypes for Mars rovers described by Shen and Nagy, and Miller et. al. [19, 21]. For example, [19] discusses strategies by which a mobile robot can determine its location after a short period of "reconnaissance". In a related vein (outdoor navigation in less exotic landscapes) Thompson et. al. [23] also describe the role of "reconnaissance" in the localization problem.

Despite the considerable attention it has received in the robotics literature, localization has been the subject of very little theoretical work. Here we consider an environment $\mathcal{E}$, which for the purposes of this discussion can be taken to 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. 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 equivalent 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, all consistent with its current view. In what has been essentially the only previous theoretical paper on the problem, Guibas, Motwani, and Raghavan [11] 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 an efficient algorithm for this enumeration problem, as well as techniques for pre-processing $\mathcal{E}$ to answer repeated "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 the 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.*) This building of larger and larger partial maps is precisely the notion of "reconnaissance" discussed above.

In this work, therefore, we address the following question: given a map of an environment $\mathcal{E}$, how should a robot move around in $\mathcal{E}$ so as to determine its location while traveling as little as possible?

We argue that the localization problem is a very natural setting in which to apply the competitive analysis of on-line algorithms (cf. Sleator and Tarjan [22]), first introduced into the realm of navigation problems by Papadimitriou and Yannakakis [20]. For our purposes here, 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 — for a robot running some given localization algorithm, we can then compare the distance it travels to the length of the shortest possible localizing path. In this sense, localization becomes an on-line problem: the distance the robot travels before localizing is compared to the length of the optimum solution. 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 $x \in \mathcal{E}$, is its *competitive ratio.* (An algorithm with a competitive ratio at most c will sometimes be described as c-competitive.)

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

1. As already discussed, localization is a problem that arises in a number of diverse robotics applications.

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.

Our Results

The only previous theoretical treatment of the localization problem is the paper of Guibas, Motwani, and Raghavan [11], and as mentioned above, it deals only with the "static" version of the question. Our work represents, to our knowledge, the first use of the competitive ratio in analyzing the the localization problem for mobile robots.

In what follows, we deal with the following types of environments.

- Bounded-degree trees embedded in $\Re^d$. That is, the vertices are realized by points in $\Re^d$ 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, consisting of n rectangles [2, 3, 6, 20]. 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. (So the rectangles cannot be "stuck together" to make more complex obstacles.) Although not crucial, we will also assume for the sake of uniformity that the n rectangles are surrounded by a large polygonal boundary — that is, the robot is always constrained to move inside this polygon.

We give a localization algorithm for trees with competitive ratio $O(n^{2/3})$, and a localization algorithm for rectangle packings with competitive ratio at most $O(n\sqrt{\frac{\log\log n}{\log n}}) = o(n)$. In the other direction, we show that no on-line localization algorithm can be better than $\Omega(\sqrt{n})$-competitive for either of these types of environments. Closing this gap between upper and lower bounds remains an interesting open question.

Perhaps the best way to view these algorithms is as an asymptotic improvement on *spiral search.* Indeed, arguably the most natural way to design a localization algorithm for either of these environments is via the spiral search algorithm of Baeza-Yates, Culberson, and Rawlins [1]. This natural technique has found applications, whether explicitly or implicitly, in numerous on-line navigation problems [1, 6, 12, 14, 15, 20], layered graph traversal [10, 20], the design of hybrid algorithms [13], and even the approximation of NP-hard optimization problems [5, 24]. For our purposes here, spiral search would 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 the types of environments described above. In a number of navigation problems based on exploration and search, it is often not difficult to show that this approach is optimal up to constant factors. Indeed a great deal of work is often done to determine what these constant factors are [1, 13, 14]. One consequence of our algorithms is that, in contrast to these examples, spiral search is not the best one can do for localization.

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. One other recent example of this is the k-trip shortest-path problem considered by Blum and Chalasani [4], where a robot must make repeated trips between the same pair of points, and can thus make use of partial maps for the 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" [8].

In the case of localization, knowledge of the map allows the robot to begin focusing its search as it sees more and more of its surroundings. Indeed, our algorithm is quite natural and simple to state; the main difficulty is 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.

This paper is organized as follows. In Sections 2 and 3 we give the algorithm for trees and rectangle packings respectively. In Section 4, we present constructions showing an $\Omega(\sqrt{n})$ lower bound on the best competitive ratio attainable by any on-line algorithm for this problem, in both types of environments considered here. Finally, in Section 5, we consider the problem of placing unique landmarks in the environment to make the robot's task easier.

2 The Algorithm for Trees

Here we show an $O(n^{2/3})$-competitive algorithm for localization in an arbitrary geometric tree. Recall that by a geometric tree T we mean a pair (V, E), where V is a finite point set in $\Re^d$ 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. 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. Again changing notation slightly, we assume 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 $U \subset V$, let $T(U)$ denote the subtree induced by U. When $U = \{x, y\}$, this is simply the path from x to y; its length (as a polygonal path under the Euclidean distance) will be denoted $d_T(x, y)$. Recall that the robot is constrained to move on the vertices and edges of T; we also assume it can make no use of vision other than to know the orientation of all edges incident to its current location. Finally, 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 last assumptions can be avoided at the cost of more cumbersome definitions below.)

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

Theorem 1 *There is an algorithm which is $O(n^{2/3})$-competitive for localization on any geometric tree.*

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 because the robot can initially perform a two-way spiral search to reach the closest vertex, traveling no more than 9 times too far [1]. Moreover, we can 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 1.

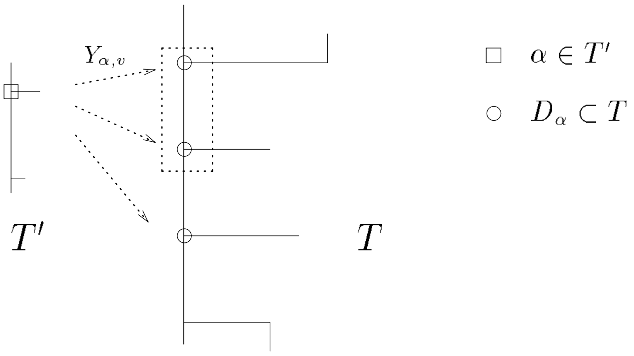

Figure 1: Lining up T' with T

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 1 *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.

(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 1) 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 $w(T')$ origins.

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

```
Initially b(T') = 1, p(T') = O(n),
  and 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 then 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 [1]. 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 $r(T') = 2^j$ of $Z(\gamma_0)$, for $j = 0, 1, 2, \ldots$. At any given time, we know the robot failed to localize after seeing *all* points within $r(T')/2 = 2^{j-1}$ of $Z(\gamma_0)$ (in the previous iteration); thus $r(T')/2$ is a lower bound on the length of the optimal solution.

Lemma 2 *In Step 1, the robot travels 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 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

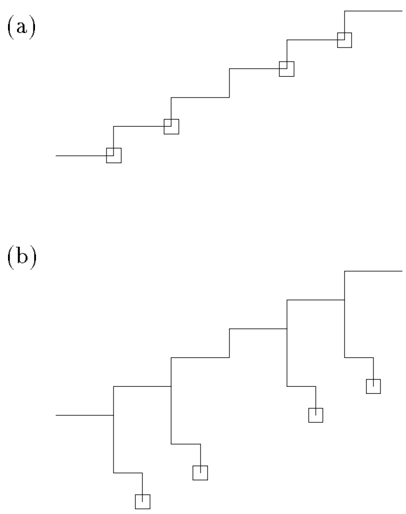

Figure 2: (a) A periodic path. (b) A comb tree.

$T(u_i, v_i)$ (the *teeth*), all of which are mutually congruent; the vertex v_i will be called the *support point* of u_i. See Figure 2.

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 $w(T')$ origins in T. We will use W to denote this set $(D_\alpha \cap Y_{\alpha,v}(T'))$ of covered origins.

Lemma 3 *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.

Claim 1 *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 2 *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 $1 < i < j \le k$. 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 1. This in turn shows that W and the distinguished point w_1 satisfy the hypotheses of Claim 2, 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 1, 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 $|D_{\alpha'} \cap Y_{\alpha',v'}| \ge |W| \ge w(T')$. Thus, by using α' instead of α, we obtain a set W' (satisfying $|W'| \ge w(T')$) which induces a periodic path. We state this as the following extension of the previous lemma.

Lemma 4 *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 5 *In 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 $m \ge w(T')$, 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 before. ■

Lemma 6 *At the end of Step 2, $p(T') \le 2n/w'(T')$.*

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 7 *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. It is important to note, first of all, that a variation of the tree constructed in Section 4 shows that the algorithm which uses only Steps 1 and 3 (i.e. Step 1 until $p(T') \leq b(T')$) 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 8 *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) with $v \in D_\alpha$. Fix such a pair (α, v); it contributes once to $w_{\alpha,v}(T')$, and for every other $\beta \in E_v$ there is a $u \in T$ such that it contributes once to $w_{\alpha,u}(T')$. Thus (α, v) contributes a total of e_v to the sum S, and hence

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

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

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

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

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

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 2 and 7, 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 occurring when $b(T')$ and $n/w(T')$ are at most $n^{2/3}$. Thus by Lemmas 2 and 5, it travels at most $O(n^{2/3})$ times too far in Steps 1 and 2. Now, Lemma 6 implies that it will begin Step 3 with $p(T') \leq 2n^{2/3}$, so by Lemma 7, 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 1.

3 The Algorithm for Rectangles

Again, to keep complications related to visibility to a minimum, we work with a rectangle packing [2, 3, 6, 20], 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 3 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 2; rather, the robot waits until the spiral-search branching factor becomes too large, and then begins exploring *several* critical paths in succession. As before, the algorithm that uses only Steps 1 and 3 is no better than $O(n)$-competitive.

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') = O(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

For $j = 0, 1, 2, \ldots$, the robot iteratively explores all parts of the environment that are within a distance of 2^j of its starting point γ_0. In order to implement this, the robot must know the shortest-path distance from γ_0 to each point as it reaches it; fortunately, there is a "compact search" subroutine due to Betke, Rivest, and Singh [3] 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 [3] — the robot turns back when it is about to move more than 2^j from γ_0.

Lemma 9 *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 [3] 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 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 2, the robot, starting from $Z(\alpha)$, 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 10 *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 11 *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 10, 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

$$d_R(v, v') \leq \frac{2r(R')}{\lambda(n)^2}. \qquad (1)$$

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 (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 distance of at most $\frac{4n}{\lambda(n)} \cdot r(R')$ in all short iterations.

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 6, 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. Moreover, 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. ∎

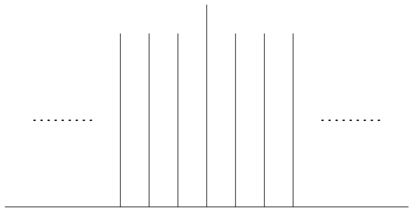

Figure 3: A lower bound for localization

Step 3 is implemented just as before. Thus, by Lemmas 9, 11, and the analogue of Lemma 7 for rectangles, we have

Theorem 2 *The above algorithm is* $O(\frac{n}{\lambda(n)})$ *- competitive for localization in an environment of* n *rectangles.*

4 Lower Bounds

Here we give an n-leaf tree embedded in the plane, for which no on-line localization algorithm (deterministic or randomized) can be better than $\Omega(\sqrt{n})$-competitive. The same construction can be modified to give an $\Omega(\sqrt{n})$ lower bound for rectangle packings; the details are left to the reader. The tree T_h is shown in Figure 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.

First we argue that if $h = \lfloor \sqrt{n} \rfloor$, then no deterministic 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. It is easy to adapt these arguments to obtain an $\Omega(\sqrt{n})$ lower bound for randomized algorithms, though with a somewhat smaller constant.

It is important to note, however, that 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 $O(n)$.

Finally, observe that this tree with $h = n$ shows that the algorithm which (in the terminology of the previous two sections) uses only Steps 1 and 3, without making use of Step 2, is no better than $O(n)$-competitive.

5 Placing Unique Landmarks

Until now, we have been considering the "drop-off" version of the problem [23], 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 visits a vertex v, it can determine the value $\mu(v)$. The goal here is to give a k-marking of $\mathcal{E}$ and an accompanying localization algorithm with the lowest possible competitive ratio.

The problem of placing k landmarks optimally in a given environment appears to be difficult, and we leave it as an open question. Here, we consider statements that can be made in general for trees and rectangles.

Proposition 1 *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 [17], 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 [17] 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. ■

For rectangular obstacles, we prove a similar tradeoff non-constructively.

Proposition 2 *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 simply be to perform spiral search as in Step 1 of Section 3 until $p(T')$ decreases to 1 or a landmark is reached (at which point $p(R')$ decreases to 1 immediately). 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 9. ∎

6 Conclusion and Open Problems

We have presented the first competitive analysis of the localization problem for mobile robots, and have given algorithms which improve asymptotically on the spiral search technique. As noted earlier, the building blocks of these algorithms are quite natural, and we believe that they provide some insight into the structure of the localization problem. There are a number of interesting questions left open by this work; two natural directions for further work are to narrow the gaps between the upper and lower bounds, and to determine the algorithmic complexity of placing landmarks optimally in an environment.

Acknowledgements

The mobile robot localization problem was initially mentioned to the author by Leo Guibas; the specific problems considered here grew out of discussions with Dan Huttenlocher at Xerox PARC. Thanks also to Michel Goemans for his comments.

References

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

[2] 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.

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

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

[5] 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.

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

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

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

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

[10] 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.

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

[12] B. Kalyanasundaram, K. Pruhs, "A competitive analysis of nearest neighbor algorithms for searching unknown scenes," *Proc. 9th Annual Symposium on Theoretical Aspects of Computer Science*, 1992, pp. 147–157.

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

[14] 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.

[15] J. Kleinberg, "On-line search in a simple polygon," *Proc. 5th ACM-SIAM Symposium on Dis-*

crete Algorithms, 1994.

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

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

[18] M. Manasse, L. McGeoch, D. Sleator, "Competitive algorithms for on-line problems," *Proc. Twentieth ACM Symposium on Theory of Computing*, 1988, pp.322–333.

[19] 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.

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

[21] 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.

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

[23] 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.

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

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

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

Appendix

Proof of Claim 1. 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 S 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 2. 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 simply 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. Now we can 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 a collection of periodic paths. Thus, $T(U)$ is a comb tree. ■