DIMACS Technical Report 95-40
August 1995

Localizing a Robot with Minimum Travel[1]

by

Gregory Dudek[2]
Research Centre for Intelligent Machines and School of Computer Science
McGill University, Montréal, Québec, Canada H3A 2A7
dudek@cim.mcgill.ca

Kathleen Romanik[3,4]
DIMACS Center for Discrete Mathematics and Theoretical Computer Science
Rutgers University, Piscataway, NJ 08855-1179
romanik@dimacs.rutgers.edu

Sue Whitesides[5]
Research Centre for Intelligent Machines and School of Computer Science
McGill University, Montréal, Québec, Canada H3A 2A7
sue@cs.mcgill.ca

[1]An earlier version of this paper appeared as a McGill University School of Computer Science technical report SOCS-94.5 in August 1994. Also, an abridged version appeared in the Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 437-446, in January 1995.
[2]Supported by NSERC research grants programme.
[3]DIMACS University-Industry postdoctoral fellow
[4]Author also received support from IRIS National Network of Centres of Excellence and NSERC while at McGill University.
[5]Supported by NSERC research grants programme.

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

We consider the problem of localizing a robot in a known environment modeled by a simple polygon P. We assume that the robot has a map of P but is placed at an unknown location inside P. From its initial location, the robot sees a set of points called the visibility polygon V of its location. In general, sensing at a single point will not suffice to uniquely localize the robot, since the set H of points in P with visibility polygon V may have more than one element. Hence, the robot must move around and use range sensing and a compass to determine its position (i.e. localize itself). We seek a strategy that minimizes the distance the robot travels to determine its exact location.

We show that the problem of localizing a robot with minimum travel is NP-hard. We then give a polynomial time approximation scheme that causes the robot to travel a distance of at most $(k-1)d$, where $k = |H|$ and d is the length of a minimum length tour that would allow the robot to verify its true initial location by sensing. We also show that this bound is the best possible.

1 Introduction

Numerous tasks for a mobile robot require it to have a map of its environment and knowledge of where it is located in the map. Determining the position of the robot in the environment is known as the *robot localization problem.* To date, mobile robot research that supposes the use of a map generally assumes either that the position of the robot is always known, or that it can be estimated using sensor data acquired by displacing the robot only small amounts [BR93, KMK93, TA92]. However, self-similarities between separate portions of the environment prevent a robot that has been dropped into or activated at some unknown place from uniquely determining its exact location without moving around. This motivates a search for strategies that direct the robot to travel around its environment and to collect additional sensory data [BD90, KB87, DJMW93] to deduce its exact position.

In this paper, we view the general robot localization problem as consisting of two phases: hypothesis generation and hypothesis elimination. The first phase is to determine the set H of *hypothetical locations* that are consistent with the sensing data obtained by the robot at its initial location. The second phase is to determine, in the case that H contains two or more locations (see Figure 1), which location is the true initial position of the robot; i.e. to eliminate the incorrect hypotheses.

Ideally, the robot should travel the minimum distance necessary to determine its exact location. This is because the time the robot takes to localize itself is proportional to the distance it must travel (assuming sensing and computation time are negligible in comparison). Also, the most common devices for measuring distance, and hence position, on actual mobile robots are relative measurement tools such as odometers. Therefore, they yield imperfect estimates of orientation, distance and velocity, and the errors in these estimates accumulate disastrously with successive motions [Dav86]. Our strategy is well-suited to handling the accumulation of error problem via simple recalibration, as we will point out later.

A solution to the hypothesis generation phase of robot localization has been given by Guibas, Motwani and Raghavan in [GMR92]. We describe this further in the next section, after making more precise the definitions of the two phases of robot localization. Our paper is concerned with minimizing the distance traveled in the hypothesis elimination phase of robot localization. It begins where [GMR92] left off. Together, the two papers give a solution to the general robot localization problem.

In this paper, we define a natural algorithmic variant of the problem of localizing a robot with minimum travel and show this variant is NP-hard. We then solve the hypothesis elimination phase with what we call a *greedy localization strategy.* Our strategy causes the robot to travel a distance that is bounded above by $(k-1)d$, where $k = |H|$ and d is the length of a minimum length tour that allows the robot to verify its true initial position by sensing. This is remarkable in view of the fact that the length of this minimum length tour cannot be determined without *a priori* knowledge of which hypothetical location in H is the true one, and yet our strategy determines a path whose length is provably within a factor $k-1$ of the best possible without access to such *a priori* knowledge. Thus our strategy can be seen as a $(k-1)$-competitive on-line algorithm according to the notion of competitive on-line algorithms introduced by Sleator and Tarjan [ST85], since the tour it generates is no longer than $k-1$ times the shortest tour. We also show that *any* localization strategy that does not have *a priori* knowledge of which hypothetical location in H is the true one will travel at least distance $(k-1)d$ in the worst case, so our strategy is the best possible.

The rest of this paper is organized as follows. In Section 2 we give a formal definition of the robot localization problem, we define some of the terms used in the paper, and we comment on previous work. In Section 3 we prove that given a solution set H to the hypothesis generation phase of the localization problem that contains more than one hypothetical location, the hypothesis elimination phase, which localizes the robot by using minimum travel distance, is NP-hard. In Section 4 we define the geometric structures that we use to set-up our greedy localization strategy. In Section 5 we give our greedy localization strategy and prove the previously mentioned performance guarantee of $k-1$ times optimum. Section 6 summarizes and comments on open problems.

2 Localization Through Traveling and Probing

In this section, we describe our robot abstraction and give some key definitions.

The most common application domain for mobile robots is indoor "structured" environments. In such environments it is often possible to construct a map of the environment, and it is acceptable to use a polygonal approximation P of the free space [Lat91] as a map. A common sensing method used by mobile robots is range sensing (for example, sonar sensing or laser range sensing).

2.1 Assumptions about the robot

We assume the following throughout this paper.

- The robot is mobile and moves in a static 2-dimensional obstacle-free environment. We model the movement of the robot in the environment by a point p moving inside and along the boundary of an n-vertex polygon P positioned somewhere in the plane.

- The robot has the ability to make error-free motions between arbitrary locations in the environment[1].
- The robot has a map of its environment, i.e., it knows both P and the orientation of P in the plane.
- The robot has a compass and a range sensing device. It is essential that the robot be able to determine its orientation (with the compass); otherwise it can never uniquely determine its exact location in an environment with non-trivial symmetry (such as a square).
- At any time, the robot can make a set of observations from its current location. The observations at a particular location determine a polygon V of points that the robot can see from that location (see the next subsection for a definition of V). The robot also knows its location in V.
- The robot's sensor can detect the distances to those points on walls for which an unobstructed straight line can be drawn from the robot. This is analogous to what can be extracted by various real sensors such as laser range finders.

In order to abstract the sensory interpretation process, we define a *signature* that the robot will compute for its current location, based on its observations. The particular definition of signature we will adopt is one that assures that there are only a finite number of signatures for all the points in P.

2.2 Some definitions and an example

Two points in P are *visible* to each other or *see* each other if the straight line segment joining them does not intersect the exterior of P. The *visibility polygon* $V(p)$ for a point $p \in P$ is the polygon consisting of all points in P that are visible from p. We denote by V the visibility polygon of the initial location of the robot. The data received from a range sensing device is modeled as a visibility polygon. Since there may be more than one location in P with visibility polygon V, we note that $V = V(p)$ for one or more points $p \in P$. The number of vertices of V is denoted by m. Since the robot has a compass, we assume the representations of P and V have a common reference direction.

We break the general problem of localizing a robot into two phases as follows.

The Robot Localization Problem

[1]In practice, position estimation errors accrue in the execution of such motions; however the strategy we present here is exceptionally well suited to various methods for limiting these errors using sensor feedback (see Section 5.1).

Hypothesis Generation: Given P and V, determine the set H of all points $p_i \in P$ such that the visibility polygon of p_i is exactly V (i.e. $V(p_i) = V$).

Hypothesis Elimination: Devise a strategy by which the robot can correctly eliminate all but one hypothesis from H, thereby determining its exact initial location. Ideally, the robot should travel a distance as small as possible to achieve this.

As previously mentioned, the hypothesis generation phase has been solved by Guibas, Motwani and Raghavan. We describe their results in the next subsection. This paper is concerned with the hypothesis elimination phase.

Consider the example illustrated in Figure 1. Imagine the robot knows the map polygon P of its environment and the visibility polygon V representing what it can "see" in the environment from its present location. Suppose also that it knows that P and V should be oriented as shown. The black dot represents the robot's position in the visibility polygon. By examining P and V, the robot can determine that it is at either point p_1 or point p_2 in P, i.e. $H = \{p_1, p_2\}$. It cannot distinguish between these two locations because $V(p_1) = V(p_2) = V$. However, by traveling out into the "hallway" and taking another probe, the robot can determine its location precisely.

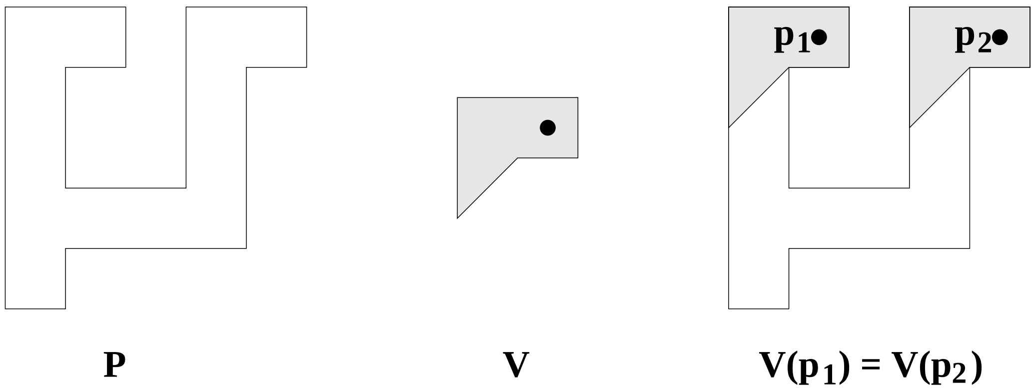

Figure 1: Given a map polygon P (left) and a visibility polygon V (center), the robot must determine which of the 2 possible initial locations p_1 and p_2 (right) is its actual location in P.

An optimal strategy for the hypothesis elimination phase would direct the robot to follow an optimal verification tour, defined as follows.

Definition. A *verification tour* is a tour along which a robot that knows its position *a priori* can travel to verify this information by probing and then return to its starting position. The tour need not be generated by any particular strategy. An *optimal verification tour* is a verification tour of minimum length d.

Since we do not assume *a priori* knowledge of which hypothetical location in H is correct, an optimal verification tour for the hypothesis elimination phase cannot be pre-computed. For this reason, we seek an interactive probing strategy to localize the robot. In each step of such a strategy, the robot uses its range sensors to compute the visibility polygon of its present position, and from this information decides where to move next to make another probe. To be precise, the type of strategy we seek can be represented by a localizing decision tree, defined as follows.

Definition. A *localizing decision tree* is a tree consisting of two kinds of nodes and two kinds of weighted edges. The nodes are either sensing nodes (S-nodes) or reducing nodes (R-nodes), and the node types alternate along any path from the root to a leaf. Thus tree edges directed down the tree either join an S-node to an R-node (SR-edges), or join an R-node to an S-node (RS-edges).

- Each S-node is associated with a position defined relative to the initial position of the robot. The robot may be instructed to probe the environment from this position.
- Each R-node is associated with a set $H' \subseteq H$ of hypothetical initial locations that have not yet been ruled out. The root is an R-node associated with H, and each leaf is an R-node associated with a singleton hypothesis set.
- Each SR-edge represents the computation that the robot does to rule out hypotheses in light of the information gathered at the S-node end of the edge. An SR-edge does not represent physical travel by the robot and hence has weight 0.
- Each RS-edge has an associated path defined relative to the initial location of the robot. This is the path along which the robot is directed to travel to reach its next sensing point. The weight of an RS-edge is the length of its associated path.

Since we want to minimize the distance traveled by the robot, we define the weighted height of a localizing decision tree as follows.

Definition. The *weight* of a root-to-leaf path in a localizing decision tree is the sum of the weights on the edges in the path. The *weighted height* of a localizing decision tree is the weight of a maximum-weight root-to-leaf path. An *optimal localizing decision tree* is a localizing decision tree of minimum weighted height.

In the next section, we show that the problem of finding an optimal localizing decision tree is NP-hard.

We call a localization strategy that can be associated with a localizing decision tree a *localizing decision tree strategy.* As an example of such a strategy, consider the map polygon P shown on the left in Figure 2.

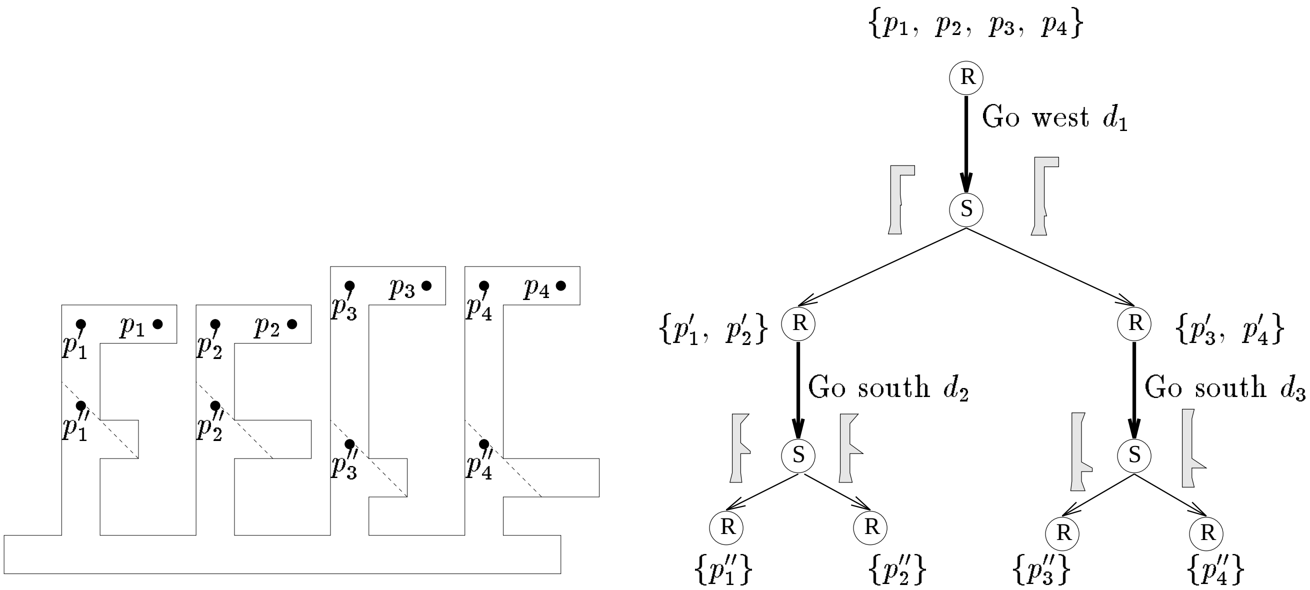

Figure 2: A map polygon and 4 hypothetical locations $\{p_1, p_2, p_3, p_4\}$ (left) with a localizing decision tree for determining the true initial position of the robot (right).

Imagine that from the visibility polygon sensed by the robot at its initial location it is determined that the set of hypothetical locations is $H = \{p_1, p_2, p_3, p_4\}$. Hence the root of the localizing decision tree (shown on the right in Figure 2) is associated with H. In the figure, the SR-edges are labeled with the visibility polygons seen by the robot at the S-node endpoints of these edges. Assuming that north points straight up, the strategy given by the tree directs the robot first to travel west a distance d_1, which is the distance between p_i and p'_i, for $1 \leq i \leq 4$. This positions the robot at one of p'_1, p'_2, p'_3 or p'_4. The strategy then directs the robot to take another probe at its new location. This is represented by an S-node in the decision tree. Depending on the outcome of the probe, the robot knows it is located either at one of $\{p'_1, p'_2\}$ or at one of $\{p'_3, p'_4\}$. If it is located at p'_1 or p'_2, then the strategy directs it to travel south a distance d_2, which is the distance between p'_1 and p''_1 (and between p'_2 and p''_2), to a position just past the dotted line segment shown in P. By taking a probe from below this line segment, it will be able to see the vertex at the end of the segment if it is at location p''_1, and it will not see this vertex if it is at location p''_2. Thus after this probe it will be able to determine its unique location in P. Similarly, if the robot is located at p'_3 or p'_4, then the strategy directs it to travel south a distance d_3 (which is the distance between p'_3 and p''_3 and also between p'_4 and p''_4) to either p''_3 or p''_4 and to take another probe, determining its initial position. The farthest that the robot must travel to determine its location is $d_1 + d_3$, so the weighted height of this decision tree is $d_1 + d_3$.

2.3 Previous work

Previous work on robot localization by Guibas, Motwani, and Raghavan in [GMR92] showed how to preprocess a map polygon P so that given the visibility polygon V that a robot sees, the set of points in P whose visibility polygon is congruent to V, and oriented the same way, can be returned quickly. Their algorithm preprocesses P in $O(n^5 \log n)$ time and $O(n^5)$ space, and it answers queries in $O(m + \log n + k)$ time, where n is the number of vertices of P, m is the number of vertices of V, and k is the size of the output (the number of places in P at which the visibility polygon is V). They also showed how to answer a single localization query in $O(mn)$ time with no preprocessing.

Kleinberg [Kle94b] has independently given an interactive strategy for localizing a robot in a known environment. As in our work, he seeks to minimize the ratio of the distance traveled by the robot using his strategy to the length of an optimal verification path. The worst-case value of this ratio over all starting points is called the *competitive ratio* of the strategy, in the sense of on-line algorithms [ST85]. If the robot travels a distance no more than k times the length of an optimal verification path, then the strategy can be called *k-competitive.*

Kleinberg's model differs from ours in several ways. First of all, he models the robot's environment as a geometric tree rather than a simple polygon. A *geometric tree* is a pair (V, E), where V is a finite point set in $\mathbb{R}^d$ and E is a set of line segments whose endpoints all lie in V. The edges E do not intersect except at points of V and do not form cycles. Kleinberg only considers geometric trees with bounded degree Δ. Also, his robot can make no use of vision other than to know the orientation of all edges incident to its current location. Using this model, Kleinberg gives an $O(n^{2/3})$-competitive algorithm for localizing a robot in a geometric tree with bounded degree Δ, where n is the number of *branch vertices* (vertices of degree greater than 2) of the tree.

The competitive ratio of Kleinberg's algorithm appears to be better than the lower bound illustrated by Figure 10 in Section 5.3. However, if this map polygon were modeled as geometric tree it would have degree n, where n is the number of branch vertices, rather than a constant degree, and the distance traveled by a robot using Kleinberg's algorithm can be linear in the degree of the tree. If Kleinberg's algorithm ran on this example, it would only execute Step 1, which performs a spiral search, and it would cause the robot to travel a distance almost $4n$ times the length of an optimal verification path. Our algorithm causes the robot to travel a distance less than $2n$ times the length of an optimal verification path on this example. Our algorithm is similar to Step 3 of Kleinberg's algorithm, and he gives a lower bound example (in Figure 3 of his paper) illustrating that an algorithm using only Steps 1 and 3 of his algorithm is no better than $O(n)$-competitive. However, this example does not apply to our model since the robot in our model has the ability to see to the end of the hallway and localize itself using its distance to the end.

Other theoretical work on localizing a robot in a known environment has also been done. Betke and Gurvits [BG94] gave an algorithm that uses the angles subtended by landmarks in the robot's environment to localize a robot. Their algorithm runs in time linear in the number of landmarks, and it assumes that a correspondence is given between each landmark seen by the robot and a point in the map of the environment. Avis and Imai [AI90] also investigated the problem of localizing a robot using angle measurements, but they did not assume any correspondence between the landmarks seen by the robot and points in the environment. Instead they assumed that the environment contains n identical markers, and the robot takes k angle measurements between an unknown subset of these markers. They gave polynomial time algorithms to determine all valid placements of the robot, both in the case where the robot has a compass and where it does not. In addition they showed that with polynomial-time preprocessing location queries can be answered in $O(\log n)$ time.

Theoretical work with a similar flavor to ours has also been done on navigating a robot in an unknown environment. In this work a point robot must navigate from a point s to a target t, which is either a point or an infinite wall, where the Euclidean distance from s to t is n. There are obstacles in the scene, which are not known *a priori*, but which the robot learns about only as it encounters them. The goal is to optimize (i.e. minimize) the ratio of the distance traveled by the robot to the length of a shortest (obstacle-free) path from s to t. As with localization strategies, this ratio is called the competitive ratio of the strategy. If a dynamic navigation strategy always reaches the target t by traveling a distance no more than k times the length of the shortest path, then the strategy is called k-competitive.

Papadimitriou and Yannakakis [PY91] gave a strategy for navigating between two points, where all obstacles are unit squares, that achieves a competitive ratio of 1.5, which they show is optimal. For squares of arbitrary size they gave a strategy achieving a ratio of $\sqrt{26}/3$. They also showed, along with Eades, Lin and Wormald [ELW93], that when t is an infinite wall and the obstacles are oriented rectangles, there is a lower bound of $\Omega(\sqrt{n})$ on the ratio achievable by any strategy.

Blum, Raghavan and Schieber [BRS91] gave a strategy that matched the $\Omega(\sqrt{n})$ lower bound for navigating between two points with oriented, rectangular obstacles. Their strategy combines strategies for navigating from a point to an infinite wall and from a point on the wall of a room to the center of the room, with competitive ratios of $O(\sqrt{n})$ and $O(2^{\sqrt{3 \log n}})$ respectively.

Several people have studied the problem of navigating from a vertex s to a vertex t inside an unknown simple polygon. They assume that at every point on its path the robot can get the visibility polygon of that point. Klein [Kle92] proved a lower bound of $\sqrt{2}$ and gave a strategy achieving a ratio of 5.72 for the class of street polygons. A *street* is a simple polygon such that the clockwise chain L and the counterclockwise chain R from s to t are mutually weakly visible. That is, every point on L is visible to some point on R and visa versa. Kleinberg [Kle94a] gave a strategy that improved Klein's ratio to $2\sqrt{2}$, and Datta and Icking [DI94] gave a strategy with a ratio of 9.06 for a more general class of polygons called *generalized streets*, where every point on the boundary is visible from a point on a horizontal line segment joining L and R. They also showed a lower bound of 9 for this class of polygons.

Previous work in the area of geometric probing has examined the complexity of constructing minimum height decision trees to uniquely identify one of a library of polygons in the plane using point probes. Such probes examine a single point in the plane to determine if an object is located at that point. If each polygon in the library is given a fixed position, orientation and scale, then it has been shown that both the problem of finding a minimum cardinality probe set (for a noninteractive probing strategy) [BS91] and the problem of constructing a minimum height decision tree for probing (for an interactive strategy) [AMM$^+$93] are NP-Complete. Arkin et al.[AMM$^+$93] give a greedy strategy that builds a decision tree of height at most $\lceil \log k \rceil$ times that of an optimal decision tree, where k is the number of polygons in the library. The minimum height decision tree used for probing in [AMM$^+$93] is different than our localizing decision tree. It is a binary decision tree whose internal nodes represent point probes whose outcome is either positive or negative and whose edges are unweighted. The height of such a decision tree is just the number of levels of the tree, and it represents the maximum number of probes necessary to identify any polygon in the library.

3 Hardness of Localization

In this section we show that the problem of constructing an optimal localizing decision tree, as defined in the previous section, is NP-hard. To do this, we first formulate the problem as a decision problem.

ROBOT-LOCALIZING DECISION TREE (RLDT)

INSTANCE: A simple polygon P and a star-shaped polygon V, both with a common reference direction, the set H of all locations $p_i \in P$ such that $V(p_i) = V$, and a positive integer h.

QUESTION: Does there exist a localizing decision tree of weighted height less than or equal to h that localizes a robot with initial visibility polygon V in the map polygon P? In other words, does there exist a localizing decision tree of weighted height less than or equal to h that determines which of the possible initial locations in H is the true location of a robot, given that the initial visibility polygon of the robot is V?

We show that this problem is NP-hard by giving a reduction from the ABSTRACT DECISION TREE problem, proven NP-complete by Hyafil and Rivest in [HR76]. The ABSTRACT DECISION TREE problem is stated as follows:

Abstract Decision Tree (ADT)

INSTANCE: A set $X = \{x_1, \ldots, x_k\}$ of objects, a set $\mathcal{T} = \{T_1, \ldots, T_n\}$ of subsets of X representing binary tests, where test T_j is positive on object x_i if $x_i \in T_j$ and is negative otherwise, and a positive integer $h' \leq n$.

QUESTION: Does there exist an abstract decision tree of height less than or equal to h', where the height of a tree is the maximum number of edges on a path from the root to a leaf, that can be constructed to identify the objects in X? Such a decision tree has a binary test at all internal nodes and an object at every leaf. To identify an unknown object, the test at the root is performed on the object, and if it is positive the right branch is taken, otherwise the left branch is taken. This procedure is repeated until a leaf is reached, which identifies the unknown object.

Theorem 1 *RLDT is NP-hard.*

Proof: Given an instance of ADT, we create an instance of RLDT as follows. We construct P to be a staircase polygon, with a stairstep for each object $x_i \in X$ (see Figure 3). For each stairstep we construct $n = |\mathcal{T}|$ protrusions, one for each test in $\mathcal{T}$ (see Figure 4). If test T_j is a positive test for object x_i, then protrusion T_j on stairstep x_i has an extra hook on its end (such as T_3, T_4, and T_n in Figure 4). The length of a protrusion is denoted by l and the distance between protrusions T_1 and T_n is denoted by d, where d and l are chosen so that $dh' < l$. The vertical piece between adjacent stairsteps is longer than $(2l + d)h'$, and the width w of each stairstep is much smaller than the other measurements. The polygon P has $O(nk)$ vertices, where $n = |\mathcal{T}|$ and $k = |X|$.

Consider a robot that is initially located at the shaded circle shown in Figure 4 on one of the k stairsteps. The visibility polygon V at this point has $O(n)$ vertices and is the same at an analogous point on any internal stairstep x_i. We output the polygons P and V, which can be constructed in polynomial time, the k locations $p_i \in P$ such that $V(p_i) = V$, and weighted height $h = (2l + d)h'$ as an instance of RLDT.

In order for the robot to localize itself, it must either travel to one of the "ends" of P (either the top or the bottom stairstep) to discover on which stairstep it was located initially, or it must examine a sufficient number of the n protrusions on the stairstep where it is located to distinguish that stairstep from all the others. Since the vertical piece of each stairstep is longer than $h = (2l + d)h'$, only a strategy that directs the robot to remain on the same stairstep can lead to a decision tree of weighted height $\leq h$.

Any decision tree that localizes the robot by examining protrusions on the stairstep corresponds to an equivalent abstract decision tree to identify the objects of X using tests in $\mathcal{T}$. Each time the robot travels to the end of protrusion T_j to see if it has an extra hook

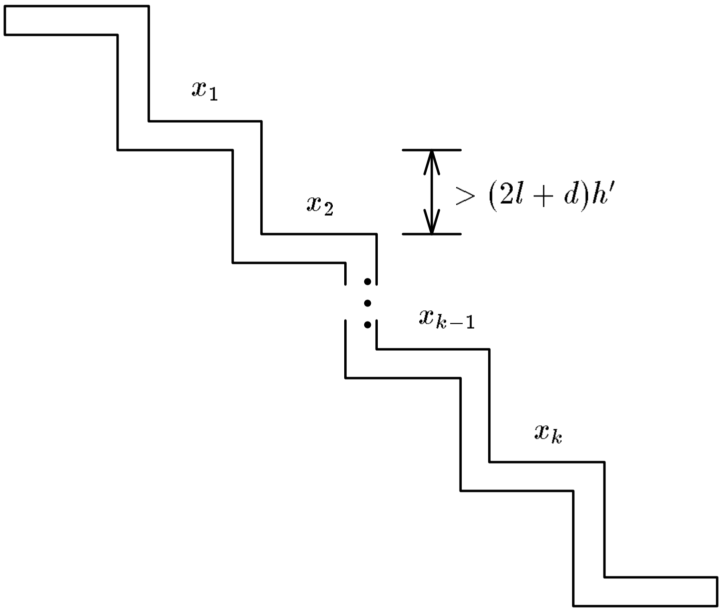

Figure 3: Construction showing localization is NP-hard.

on its end, it corresponds to performing binary test T_j on an unknown object to observe the outcome. The robot must travel $2l$ to "perform" this test (since the robot is a point and w is small, we can assume that it starts exactly at the bottom of the protrusion), and it travels at most d in between tests. Therefore, if the robot can always localize itself by examining no more than h' protrusions, then it has a decision tree of weighted height no more than $h = (2l + d)h'$, which corresponds to an abstract decision tree of height h' for the ADT problem. Since $dh' < l$, in a localizing decision tree of weighted height $\leq h$ the robot cannot examine more than h' protrusions on any root-to-leaf path. Similarly, any abstract decision tree to identify the objects of X using tests in $\mathcal{T}$ corresponds to an equivalent decision tree

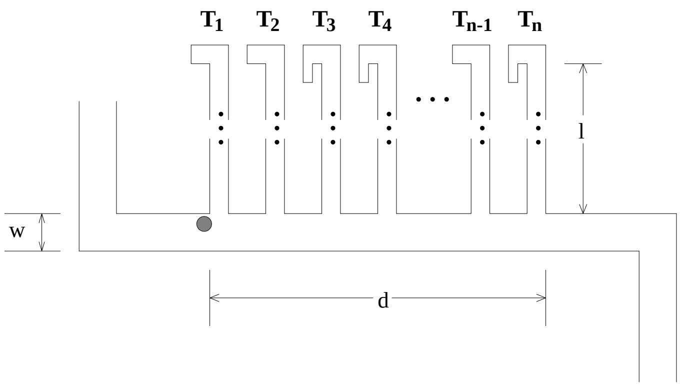

Figure 4: Close-up of a stairstep x_i in NP-hard construction. Not to scale: $l >> d >> w$.

to localize the robot by examining protrusions. □

4 Using a Visibility Cell Decomposition for Localization

In this section we discuss the geometric issues involved in building a data structure for our greedy localization strategy.

4.1 Visibility cells and the overlay arrangement

When we consider positions where the robot can move to localize itself, we reduce the infinite number of locations in P to a finite number by first creating a visibility cell decomposition of P [Bos91, BLM92, GMR92].

A *visibility cell* (or *visibility region*) C of P is a maximally connected subset of P with the property that any two points in C see the same subset of vertices of P ([Bos91, BLM92]). A *visibility cell decomposition* of P is simply a subdivision of P into visibility cells. This decomposition can be computed in $O(n^3 \log n)$ using techniques in [Bos91, BLM92]. It is created by introducing $O(nr)$ lines, called *visibility edges*, into the interior of P, where r is the number of reflex vertices[2] of P. Each line starts at a reflex vertex u, ends at the boundary of P, and is collinear with a vertex v that is either visible from u or is adjacent to it. The number of cells in this decomposition, as well as their total complexity, is $O(n^2 r)$ (see [GMR92] for a proof).

Although two points p and q in the same visibility cell C see the same subset of vertices of P, they may not have the same visibility polygon (i.e. it may be that $V(p) \neq V(q)$). This is because some edges of $V(p)$ may not actually lie on the boundary of P (these edges are collinear with p and are produced by visibility lines), so these edges may be different in $V(q)$. Therefore, in order to represent the portion of P visible to a point p in a visibility cell C in such a way that all points in C are equivalent, we need a different structure than the visibility polygon. The structure that we use is the *visibility skeleton* of p.

Definition. The *visibility skeleton* $V^*(p)$ of a location $p \in P$ is the skeleton of the visibility polygon $V(p)$. That is, it is the polygon induced by the non-spurious vertices of $V(p)$, where a *spurious* vertex of $V(p)$ is one that lies on an edge of $V(p)$ that is collinear with p, and the other endpoint of this edge is closer to p. The edges of the skeleton are labeled to indicate

[2] A *reflex vertex* of P is a vertex that subtends an angle greater than 180°.

which ones correspond to real edges from P and which ones are *artificial* edges induced by the spurious vertices. If p is outside P, then $V^*(p)$ is equal to the special symbol $\emptyset$.

For a complete discussion of visibility skeletons and a proof that $V^*(p) = V^*(q)$ for any two points p and q in the same visibility cell, see [Bos91, BLM92, GMR92]. Because points in the same visibility cell have the same visibility skeleton, and there are only $O(n^2r)$ visibility cells, we use this for the signature of a location.

Choice of signature
The *signature* associated with a location $p \in P$ is its visibility skeleton $V^*(p)$.

As stated in Section 2, the hypothesis generation phase of the robot localization problem generates a set $H = \{p_1, p_2, \dots p_k\} \subset P$ of *hypothetical locations* at which the robot might be located initially. The number k of such locations is bounded above by r (see [GMR92] for a proof). From this set H, we can select the first location p_1 (or any arbitrary location) to serve as an origin for a local coordinate system. For each location p_j, $1 \leq j \leq k$, we define the *translation vector* $t_j = p_j - p_1$ that translates location p_j to location p_1, and we define P_j to be the *translate* of P by vector t_j. That is, P_j is the translate of P in which p_j and p_1 are coincident (note that $P_1 = P$). We thus have a set $\{P_1, P_2, \dots P_k\}$ of translates of P corresponding to the set H of hypothetical locations. Note that in each P_j, the point corresponding to the hypothetical location p_j is located at the origin.

In order to determine the hypothetical location corresponding to the true initial location of the robot, we construct an *overlay arrangement* A that combines the k translates P_j that correspond to the hypothetical locations, together with their visibility cell decompositions. More formally, we define A as follows.

Definition. The *overlay arrangement* A for the map polygon P corresponding to the set of hypothetical locations H is obtained by taking the union of the edges of each translate P_j as well as the visibility edges in the visibility cell decomposition of P_j.

See Figure 5 for an example of an overlay arrangement. Since each visibility cell decomposition is created from $O(nr)$ lines introduced in the interior of P_j, a bound on the total number of cells in the overlay arrangement as well as their total complexity is $O(k^2n^2r^2)$, which may be $O(n^6)$.

Figure 6 shows a map polygon P whose corresponding overlay arrangement for the visibility polygon shown in Figure 7(a) has $\Omega(n^5)$ cells. This polygon has a long horizontal "hallway" with k identical, equally spaced "rooms" on the bottom side of it (in Figure 6 $k = 4$). Each room has width 1 unit, and the distance between rooms is $2k - 1$ units. If the robot is far enough inside one of these rooms so that it cannot see any of the rooms on the top of the hallway, then its visibility polygon is the same no matter which room it is in. The $k - 1$ rooms on the top side of the hallway are identical, have width 1 unit, and are spaced

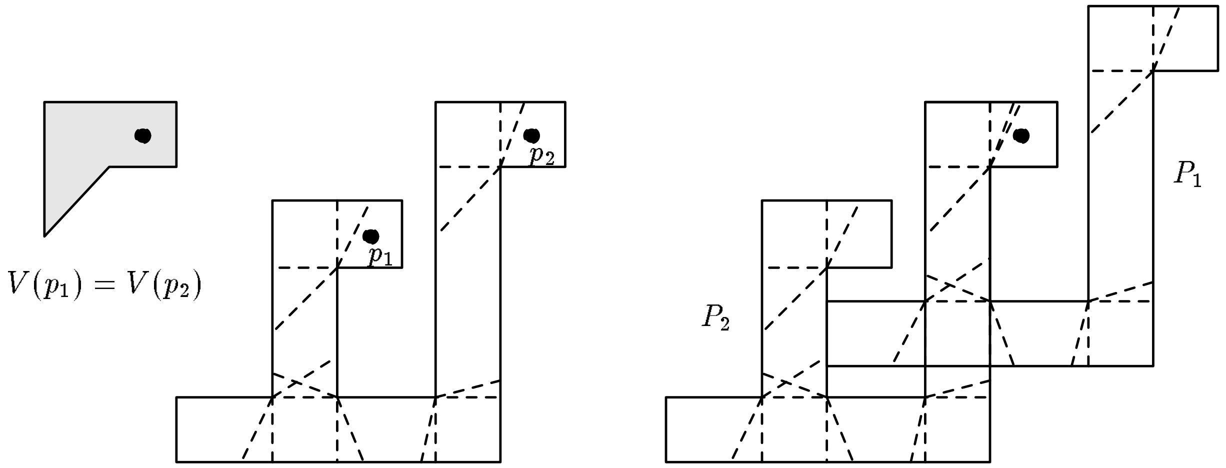

Figure 5: A visibility polygon, a map polygon and the corresponding overlay arrangement.

$2k+1$ units apart. Each top room is between two bottom rooms. The i^{th} top room from the left has its left edge a distance $2i-1$ to the right of the right edge of the bottom room to its left, and it has its right edge a distance $2(k-i)-1$ to the left of the left edge of the bottom room to its right (see Figure 6).

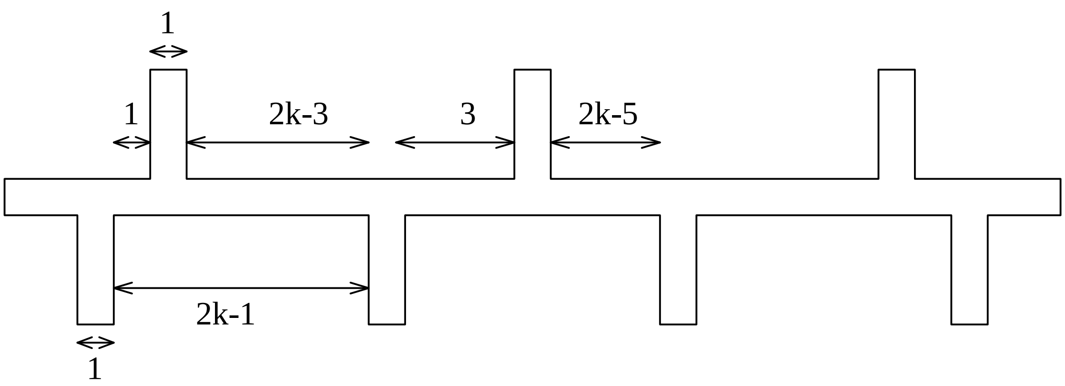

Figure 6: A map polygon whose overlay arrangement contains $\Omega(n^5)$ cells.

Consider the visibility edges starting from the reflex vertices of the bottom rooms that are generated by (i.e. collinear with) the reflex vertices of the top rooms. The i^{th} bottom room from the left will have $2(k-i)$ such visibility edges starting from its right reflex vertex and $2(i-1)$ starting from its left reflex vertex. Due to the spacing of the top rooms, the visibility edges starting from the reflex vertices of one bottom room will be at different angles than those in any other bottom room. See the picture in Figure 7(b) for an illustration of the visibility cells inside a bottom room.

When the overlay arrangement A for the visibility polygon shown in Figure 7(a) is con-

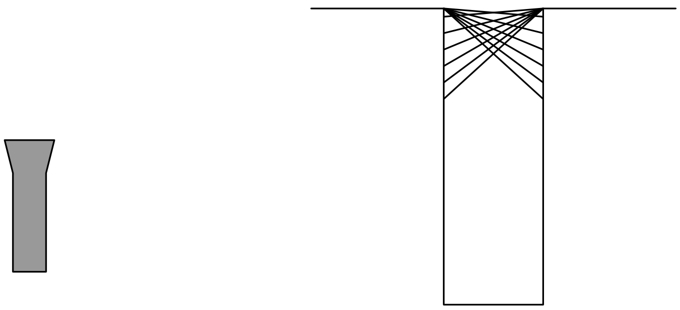

Figure 7: (a) A visibility polygon (b) Visibility cells in a bottom room

structed, it will consist of k translates, one for each of the bottom rooms of P. Since these rooms are identical and equally spaced, A will have $2k-1$ rooms on its bottom side. Since the visibility edges inside one bottom room are at different angles from those inside another bottom room, these edges will not coincide when bottom rooms from two different translates overlap in A. This means that A will have $\Omega(k)$ bottom rooms with $\Omega(k^2)$ visibility edges starting from the left reflex vertex, and $\Omega(k^2)$ visibility edges starting from the right reflex vertex, resulting in $\Omega(k^4)$ cells inside each of these bottom rooms of A. Therefore, A will have $\Omega(k^5)$ cells in total. Since the number of vertices of P is $8k$, A has $\Omega(n^5)$ cells.

Closing the gap between the upper and lower bounds on the size of the arrangement is an open problem.

4.2 The reference point set Q

Each cell in the overlay arrangement A represents a potential probe position, which can be used to distinguish between different hypothetical locations of the robot. For each cell C of A and for each translate P_j that contains C, there is an associated visibility skeleton $V_j^*(C)$. If two translates P_i and P_j have different skeletons for cell C, or if C is outside of exactly one of P_i and P_j, then C distinguishes hypothetical location p_i from p_j.

For our localization strategy we choose a set Q of *reference points* in A that will be used to distinguish between different hypothetical locations. For each cell C in A that lies in at least one translate of P, and for each translate P_j that contains C, let $q_{C,j}$ denote the point on the boundary of C that is closest to the origin (recall that translating P to P_j moves p_j to the origin). Here, the distance $d_j(q_{C,j})$ from the origin to the closest point in C is measured inside P_j. We choose $Q = \{q_{C,j}\}$. In the remainder of this paper we drop the subscripts from $q_{C,j}$ when they are not necessary.

Computing the reference points in Q involves computing Euclidean shortest paths in P_j from the origin to each cell C. To compute these paths we can use existing algorithms in the literature for shortest paths in simple polygons. We first compute for each hypothetical initial location p_j the shortest path tree from the origin to all of the vertices of P_j using the algorithm given in [GHL+87]. Since a simple polygon can be triangulated in linear time [Cha90], the shortest path tree for each p_j can also be computed in linear time. The algorithm in [GHL+87] also gives a data-structure for storing the shortest path tree so that the length of the shortest path from the origin to any point $x \in P_j$ can be found in time $O(\log n)$ and the path from the origin to x can be found in time $O(\log n + l)$, where l is the number of segments along this path. We use $\pi(p_j, x)$ to denote the shortest path from the origin to x in P_j.

To find the shortest path from the origin to a segment $\overline{xy}$ contained in P_j we use the following theorem.

Theorem 2 *If P is a simple polygon, then the Euclidean shortest path $\pi(s, \overline{xy})$ from a point s in P to a line segment $\overline{xy}$ in P is either the shortest path $\pi(s, x)$ from s to x, the shortest path $\pi(s, y)$ from s to y, or a polygonal path with l edges such that the first $l-1$ edges are the first $l-1$ edges on either $\pi(s, x)$ or $\pi(s, y)$ and the last edge is perpendicular to $\overline{xy}$.*

Proof: It is shown in [LP84] that the shortest paths $\pi(s, x)$ and $\pi(s, y)$ are polygonal paths whose interior vertices are vertices of P, and that the union of these two paths (in fact, the union of any finite number of shortest paths from s to points in P) forms a planar tree T_s rooted at s. Furthermore, it is shown that if v is the least common ancestor of x and y in T_s, then $\pi(v, x)$ and $\pi(v, y)$ are both *outward-convex* (i.e. the convex hull of each of these subpaths lies outside the region bounded by $\pi(v, x)$, $\pi(v, y)$ and the segment $\overline{xy}$). As in [LP84] we call the union $\pi(v, x) \cup \pi(v, y)$ the *funnel* associated with $\overline{xy}$, and we call v the *cusp* of the funnel. See Figure 8 for an example of a simple polygon with edges of this funnel shown as dashed line segments.

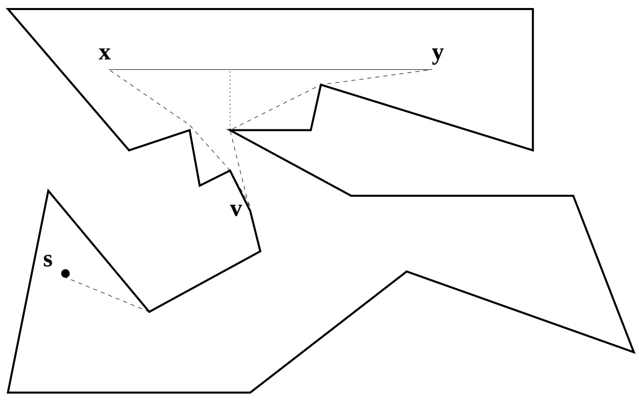

Figure 8: A simple polygon with shortest paths from s to x, y and $\overline{xy}$ shown.

The shortest path $\pi(s, \overline{xy})$ has $\pi(s, v)$ as its initial subpath. Suppose not, and let z be the point on $\overline{xy}$ where $\pi(s, \overline{xy})$ ends. Then $\pi(s, \overline{xy})$ is also the shortest path $\pi(s, z)$. Note that $z \neq x$ and $z \neq y$ or else there would be two different shortest paths to one of these points, which is a contradiction. At some point the path $\pi(s, z)$ must enter the funnel with cusp v, thereby crossing either $\pi(s, x)$ or $\pi(s, y)$. But this contradicts the fact that the union of these paths forms a planar tree.

To complete the shortest path $\pi(s, \overline{xy})$ we must find a shortest path $\pi(v, \overline{xy})$. If v has a perpendicular line of sight to $\overline{xy}$, then this visibility line will be $\pi(v, \overline{xy})$. If v does not have a perpendicular line of sight to $\overline{xy}$, then consider the edge e adjacent to v on the funnel that is the closest to perpendicular. Without loss of generality, assume e is the first edge on $\pi(v, y)$. The path $\pi(v, \overline{xy})$ will follow $\pi(v, y)$ until it reaches y or it reaches a vertex that has a perpendicular line of sight to $\overline{xy}$.

To see this, consider extending the edge e towards $\overline{xy}$ until it intersects $\overline{xy}$ at some point z. If $z \neq y$, then $\angle vzy$ is less than $90°$. Similarly, every other edge on $\pi(v, y)$ can be extended to $\overline{xy}$, and the angle that it forms with respect to y can be determined. Since the chain $\pi(v, y)$ is outward-convex, these angles will increase as we move down the chain. When the angle becomes greater than $90°$, a perpendicular line of sight exists from the endpoints of the edge. □

Using this theorem we can in $O(n)$ time determine the length of the shortest path in P_j from the origin o to $\overline{xy}$ and the closest point on $\overline{xy}$ to o. We first use the data-structure in [GHL$^+$87] to determine in $O(\log n)$ time the length d_x and the last edge e_x on the shortest path $\pi(o, x)$ and the length d_y and the last edge e_y on the shortest path $\pi(o, y)$. For each of these edges we check its angle with respect to $\overline{xy}$. Note that both of these angles cannot be $90°$ or greater, or else it would be impossible to form a funnel with $\pi(o, x)$ and $\pi(o, y)$. If the angle between e_x (e_y) and $\overline{xy}$ is at least $90°$, then we return d_x (d_y) as the shortest distance to $\overline{xy}$ and x (y) as the closest point on $\overline{xy}$.

If both the angles formed by e_x and e_y with $\overline{xy}$ are less than $90°$, then the last edge on the shortest path $\pi(o, \overline{xy})$ will be a perpendicular drawn from one of the vertices on the funnel associated with $\overline{xy}$. To find this edge we again use the data-structure in [GHL$^+$87] and follow the paths $\pi(o, x)$ and $\pi(o, y)$ to find the cusp v of the funnel. Since the data-structure in [GHL$^+$87] stores the shortest path tree by storing a parent pointer for each node of the tree, v can be found by tracing $\pi(o, x)$ back from x to o and marking all the vertices on the path and then tracing $\pi(o, y)$ back from y to the first marked vertex, which will be v. This can be done in time proportional to the number of edges examined, which is $O(n)$.

Next we examine the edges of the funnel in order, starting with e_x, and for each edge we calculate the angle formed by its extension with $\overline{xy}$. That is, for each edge (u, w) whose extension intersects $\overline{xy}$ at point z, we calculate the angle $\angle uzy$. As we move around the funnel these angles increase. When the angle becomes greater than $90°$, we have found the vertex from which to drop a perpendicular to $\overline{xy}$. It takes $O(n)$ time to find this vertex, and an additional $O(\log n)$ time to calculate the distance to $\overline{xy}$ and the closest point (this is the time it takes to determine the length of the shortest path from o to this vertex).

To compute the reference point $q_{C,j}$, we compute the shortest path distance in P_j from the origin to each edge of C. We then choose the smallest distance as the distance to the cell C. For each cell C we will have up to k reference points $\{q_{C,1}, \ldots, q_{C,k}\}$ and their corresponding distances $\{d_1(q_{C,1}), \ldots, d_k(q_{C,k})\}$. We define $d_j(q) = \infty$ for all points q not within P_j.

Partition of H

For each cell C we compute a *partition of H* that represents which hypothetical locations can be distinguished from one another by probing from inside C. If two translates P_i and P_j have the same visibility skeleton for cell C, then p_i and p_j are in the same subset of the partition of H corresponding to cell C. Also, if C is outside of both translates P_i and P_j, then p_i and p_j are in the same subset of the partition.

Since the visibility polygon and the visibility skeleton for a point $p \in P$ can be computed in $O(n)$ time (see [GA81]) and we can compare two visibility skeletons with m vertices in $O(m)$ time to see if they are identical, we can compute the partition of H for C in $O(kn+k^2m)$ time, where m is the maximum number of vertices on any of the k visibility skeletons.

Although there may be $O(n^6)$ cells in the overlay arrangement A, yielding up to $O(kn^6)$ reference points, we show in Section 5.4 that only $O(k^2)$ reference points are needed for our localization strategy, so we do not need to compute a partition of H for all $O(n^6)$ cells.

By keeping the shortest path tree computed by the algorithm in [GHL+87] for each hypothetical initial location p_j, we can later extract the actual shortest path to any cell C in A within any translate P_j in time $O(\log n + l)$, where l is the number of segments on the shortest path.

5 A Greedy Strategy for Localization

In this section we present a localizing decision tree strategy, called *Minimum Distance Localization Strategy* or *Strategy MDL* for short, for completing the solution of the hypothesis elimination phase of the robot localization problem. Our strategy, which has a greedy flavor, uses the set Q of reference points described in the previous section for choosing probing locations. Strategy MDL will enable the robot to localize itself by traveling distance at most $(k-1)d$, where $k = |H|$ and d is the length of an optimal verification tour.

In devising a localizing decision tree strategy, there are two main criteria to consider when deciding where the robot should make the next probe: (1) the distance to the new probe position, and (2) the information to be gained at the new probe position. It is easy to see that a strategy that only considers the second criterion can do arbitrarily worse than a strategy given by a minimum height localizing decision tree. Strategy MDL considers (2) only to the extent that it never directs the robot to make a useless probe. Nevertheless, its performance is the best possible. Although it would seem beneficial to weight each possible probe location with the amount of information that could be gained in the worst case by probing at that location, this change will not improve the worst-case behavior of Strategy MDL, as the lower bound example given in Section 5.3 illustrates.

Even a strategy that considers both the distance and the information criteria when choosing the next probe position can do poorly. For example, if the robot employs an incremental strategy that at each step tells it to travel to the closest probe location that yields some information, then a map polygon can be constructed such that in the worst case the robot will travel distance $2^k d$.

Using Strategy MDL for hypothesis elimination, a strategy for the complete robot localization problem can be obtained as follows. Preprocess the map polygon P using a method similar to that in [GMR92]. This preprocessing yields a data structure that stores for each equivalence class of visibility polygons either the location in P yielding that visibility polygon, if there is only one location, or a localizing decision tree that tells the robot how to travel to determine its true initial location.

5.1 Strategy MDL

In this subsection we present the details of Strategy MDL. Using the results of Section 4, it is possible to pre-compute Strategy MDL's entire decision tree. However, we will describe the strategy by explaining how it directs the robot to behave. This amounts to describing the nature of a root-to-leaf path in the tree. This is easier to understand than the description of an algorithm to compute the entire decision tree. Also, in practice, it may sometimes be preferable *not* to pre-compute the entire tree, but rather to compute the robot's next move on an interactive basis, as the robot carries out the strategy.

Information used by Strategy MDL

The following are used by Strategy MDL:

- The map polygon P.
- The set H generated in the hypothesis generation phase.
- The set Q of reference points defined in Section 4.2.

- For each point $q \in Q$ the distance $d_j(q)$ of q from the origin, measured within P_j.
- For each point $q \in Q$, a path $path_j(q)$ within P_j of length $d_j(q)$, which is defined by a series of relative motions that take the robot from the origin to q in P_j. Using the data structure in [GHL+87] these paths can be easily computed.
- For each point $q_{C,j} \in Q$, the partition of H associated with cell C, as defined in Section 4.2.

Next we describe how Strategy MDL directs the robot to behave. Initially, the set of hypothetical locations is the given set H. As the robot carries out the strategy, hypothetical locations are eliminated from H. Thus in our description of Strategy MDL, we abuse notation and use H to denote the shrinking set of *active hypothetical locations*; i.e. those that have not yet been ruled out. Similarly, we use Q to denote the shrinking set of *active reference points*; i.e. those that non-trivially partition the set of active hypothetical locations. We call a path $path_j(q)$ *active* if $p_j \in H$ and $q \in Q$ are both active.

Notation. Let $d_*(q*)$ denote the minimum of

$$\{\ d_j(q) \mid q \in Q \text{ and } p_j \in H \text{ are active }\}.$$

Let $path_*(q*)$ denote an active path of length $d_*(q*)$.

Strategy MDL

Strategy MDL directs the robot to travel along paths from the origin to points in the overlay arrangement A. Suppose that p_j is the true initial location of the robot. We will prove in the next subsection that Strategy MDL only directs the robot to follow paths that are contained in translate P_j. Note that a path from the origin that is contained in P_j is analogous to a path in P from location p_j.

From the initial H and Q, an initial $path_*(q*)$ can be selected. The strategy directs the robot to travel along this path and to make a probe at its endpoint. The robot then uses the information gained at the probe position to update H and Q and to determine a new $q*$ and a new $path_*(q*)$ from the origin. The strategy then directs the robot to retrace its previous path back to the origin, and then to follow the new path to its endpoint, which is the next probe location. This process stops when the size of H shrinks to 1. At this point the initial location of the robot is determined, and the robot can, if desired, be directed to return to its initial location by retracing its last path.

Note that Strategy MDL is well-suited to handling the problem of accumulation of errors caused by successive motions in the estimates of orientation, distance and velocity made by the robot's sensors . This is because the robot always returns to the origin after making a probe, so it can recalibrate its sensors.

5.2 A performance guarantee for Strategy MDL

The following theorems show that Strategy MDL directs the robot along a path whose length compares favorably with the minimum verification length d. First we show that Strategy MDL never directs the robot to pass through a wall. Then we show that Strategy MDL eliminates all hypothetical locations except the valid one, and we establish an upper bound on the length of the path produced by Strategy MDL. A corollary of Theorem 4 is that the localizing decision tree associated with Strategy MDL has a weighted height that is at most $2(k-1)$ times the weighted height of an optimal localizing decision tree.

Theorem 3 *Strategy MDL never directs the robot to pass through a wall.*

Proof: The proof is by contradiction. Suppose that p_j is the true initial location of the robot and x_j is the point on the boundary of P_j where the robot would first pass through a wall. Furthermore, suppose that when the robot attempts to pass through the wall at x_j, the path it has been directed to follow is $path_i(q)$.

Let C denote the cell of arrangement A (see Section 4.1) that contains the portion of $path_i(q)$ just before x_j. Since cell C is contained in P_j, it contributes a reference point $q_{C,j}$ to the set Q of reference points.

In order to arrive at a contradiction, it suffices to show that $q_{C,j}$ is active at the time Strategy MDL chooses $path_i(q)$ for the robot to follow. This is because $d_j(q_{C,j}) \leq d_j(x_j)$ by definition of $q_{C,j}$; furthermore, $d_j(x_j) \leq d_i(x_j)$ since the portion of $path_i(q)$ from the origin to x_j is contained within P_j, and $d_j(x_j)$ is equal to the length of a shortest path in P_j from the origin to x_j; finally, $d_i(x_j) < d_i(q)$ because x_j is an intermediate point on $path_i(q)$. Chaining these inequalities together gives $d_j(q_{C,j}) < d_i(q)$. Hence Strategy MDL would choose $path_j(q_{C,j})$ rather than $path_i(q)$ provided that $q_{C,j}$ is active at the time $path_i(q)$ is selected.

Now we show that $q_{C,j}$ is active when $path_i(q)$ is selected. Point $q_{C,j}$ is active if and only if the following two conditions hold: (1) p_j has not been eliminated from H and (2) the signature (visibility skeleton) associated with C distinguishes between at least two active hypothetical locations. Clearly condition (1) holds, since the correct hypothetical location is never eliminated from H. Condition (2) holds because the skeleton $V_j^*(C)$ associated with C relative to P_j has a real edge through the point x_j, whereas the skeleton $V_i^*(C)$ associated with C relative to P_i does not have a real edge through x_j. Therefore, the signature associated with points in C distinguishes between p_i and p_j, which are both active at the time $path_i(q)$ is chosen, so point $q_{C,j}$ is active.

This completes the proof that Strategy MDL never directs the robot to pass through a wall. $\square$

Theorem 4 *Strategy MDL localizes the robot by directing it along a path whose length is at most $(k-1)d$, where $k = |H|$ and d is the length of an optimal verification tour for the robot's initial position.*

Proof: Let p_t denote the true initial location of the robot. First we show that Strategy MDL eliminates all hypothetical initial locations in H except p_t. Suppose the contrary is true. This means that the set Q of active reference points becomes empty before the size of H shrinks to one. Let p_i be an active hypothetical initial location different from p_t at the time Q becomes empty. Translates P_i and P_t are not identical, so there is some point x_t on the boundary of P_t that does not belong to the boundary of P_i. Let C be the cell of arrangement A contained in P_t and containing x_t. C distinguishes between p_i and p_t because the skeletons associated with C relative to P_i and P_t are not the same. Therefore $q_{C,t}$ is still in the active set Q, a contradiction.

Next we establish the upper bound on the length of the path determined by Strategy MDL. Because the strategy never directs the robot to a probing site that does not eliminate one or more elements from H, it requires the robot to make a trip from its initial location to some sensing point and back at most $k-1$ times.

We claim that each round trip has length at most d. To see this, we first consider how a robot traveling along an optimal verification tour L would rule out an arbitrary incorrect hypothetical location p_b. Then we consider how Strategy MDL would rule out p_b.

Consider a robot traveling along tour L that eliminates each invalid hypothetical location at the first point x on L where the signature of x relative to the invalid hypothetical location differs from the signature of x relative to P_t. Let w be the point on L where the robot rules out p_b. The point w must lie on the boundary of some cell C in the arrangement A that distinguishes p_b from p_t. Cell C generates a reference point $q_{C,t} \in Q$, which is the closest point of C to the origin, where distance is measured inside P_t, so $d_t(q_{C,t}) \leq d_t(w)$. Since p_t is the true initial location of the robot, the distance $d_t(w)$ is equal to or less than the distance along L of w from the origin, as well as the distance along L from w back to the origin. Putting these inequalities together, we deduce that the distance $d_t(q_{C,t})$ is equal to or less than half the length of L.

Since p_t is the true initial location of the robot, it remains active throughout the entire execution of the strategy. In particular, it is active at the moment Strategy MDL directs the robot to move from the origin to the probing site where it eliminates p_b. Since p_b is about to be ruled out, it is also still active. That means that the reference point $q_{C,t}$ considered in the previous paragraph is still active, since it distinguishes p_b from p_t.

At this time Strategy MDL directs the robot to travel along $path_*(q*) = path_j(q)$. By design, the length $d_*(q*) = d_j(q)$ of this path, which is the distance the robot will travel from the origin to the next probing position, is the minimum over all $d_i(q)$ for active $p_i \in H$

and $q \in Q$. In particular, since point $q_{C,t}$ is still active, $d_*(q*)$ is equal to or less than $d_t(q_{C,t})$. But as we have already seen, this latter distance is equal to or less than half the length of L. Therefore, Strategy MDL directs the robot to travel along a loop from the origin to some probing position where the robot eliminates p_b and back, and the length of this loop is at most d. □

Following the definition of k-competitive given in Section 2.3 for robot navigation strategies, we can define a *k-competitive* robot localization strategy as one that localizes a robot by traveling a distance no more than k times the length of an optimal verification tour. By this definition Strategy MDL is $(k-1)$-competitive, where $k = |H|$. Note that if a verifying path is not required to return to its starting point, the bound for Theorem 4 becomes $2(k-1)d$.

Corollary 5 *The weighted height of the localizing decision tree constructed by Strategy MDL is at most* $2(k-1)$ *times the weighted height of an optimal localizing decision tree for the same problem.*

Proof: Consider the decision tree of Strategy MDL. Let p_h denote the initial location associated with the leaf that defines the weighted height of the tree. The weighted height of the tree is thus the distance Strategy MDL will direct the robot to travel to determine that p_h is the correct initial location, and by Theorem 4 this distance is at most $k-1$ times the minimum verification tour length for p_h. But the minimum verification tour length for p_h is at most twice the weight of a path from the root to p_h in an optimal localizing decision tree, which is at most the weighted height of the tree. The result follows from these inequalities. □

If the robot is required to return to its initial position, the bound on the weighted height of the localizing decision tree constructed by Strategy MDL drops to $k-1$.

In this paper, we do not comment further on computation time as there are many ways to implement Strategy MDL. Also, if travel times are large compared to computation times, the importance of our results is that they obtain good path lengths. The basic information needed for the analysis of a particular implementation appears in the previous section.

5.3 Lower bounds

In Corollary 5 we proved that the weighted height of the localizing decision tree built by Strategy MDL is no greater than $2(k-1)d$, where $k = |H|$ and d is the weighted height of an optimal localizing decision tree. This bound is also a lower bound for Strategy MDL.

Figure 9 illustrates a map polygon where the weighted height of the localizing decision tree of Strategy MDL is $\Omega(k)$ times the weighted height of an optimal localizing decision tree. Consider a map polygon that is a staircase polygon with $k+2$ stairs, such as the one in Figure 3, where each stairstep except the first and last one is similar to the one shown in Figure 9. Each such stairstep has k protrusions placed in a circle, with the end of each protrusion a distance d from the center of the circle. In each stairstep a different protrusion has its end extended, which uniquely identifies the stairstep. Each stairstep also has a longer protrusion, with k smaller protrusions sticking out of it. One of these smaller protrusions is extended to uniquely identify the stairstep. The first small protrusion is a distance $d+\epsilon$ from the center of the circle, and the last one is a distance $d+\epsilon+\delta$ from the center of the circle.

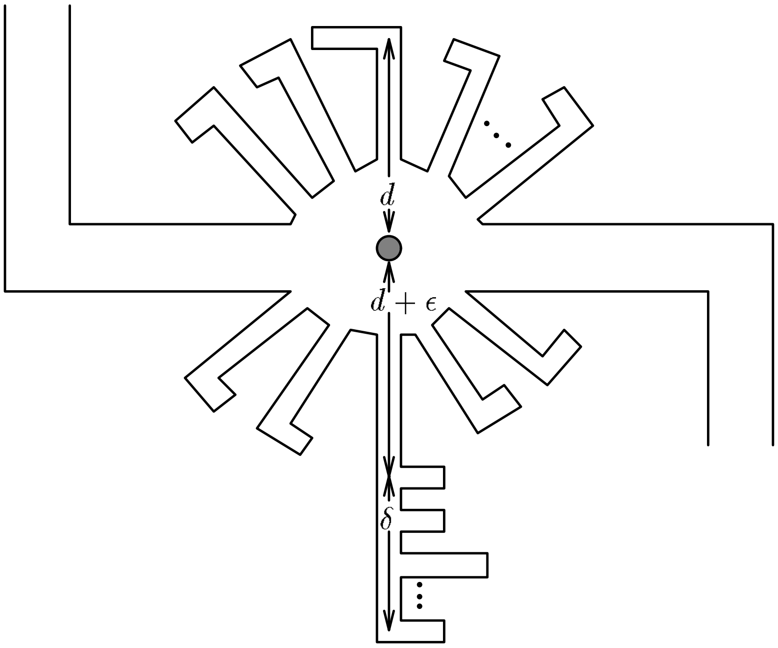

Figure 9: Part of the map polygon that gives lower bound. Not to scale: $d >> \epsilon, \delta$.

For this map polygon, if the robot is initially placed in the center of the circle on one stairstep, Strategy MDL will direct it to travel up the k protrusions of length d until it finds one that has a longer piece at the end, or until it has examined all but one of these protrusions. In the worst case the robot will travel a distance $2(k-1)d$. An optimal strategy would direct the robot to travel down the protrusion of length $d+\epsilon+\delta$ and examine all the small protrusions coming out of it until it found one that was longer. In the worst case the robot would travel a distance $d+\epsilon+\delta$. Since ϵ and δ can be made arbitrarily small, in the worst case Strategy MDL travels $\Omega(k)$ times as far as the optimal strategy. Even if we used a strategy that weighted each potential probe location with the amount of information that could be gained from that location in the worst case, we would still build the same decision tree because any probe location in the stairstep yields at most one piece of information in the worst case.

Although there are map polygons for which Strategy MDL builds a localizing decision tree whose weighted height is $\Omega(k)$ times the weighted height of an optimal localizing decision tree, there are other map polygons for which any localizing decision tree strategy builds a tree with weighted height at least $k-1$ times the length of an optimal verification tour. Consider a map polygon that is a staircase polygon with $k+2$ stairs, such as the one in Figure 3, where each stairstep except the first and last one is similar to the one shown in Figure 10. Each such stairstep has k protrusions placed in a circle, with the end of each protrusion a distance d from the center of the circle. In each stairstep a different protrusion has its end extended, which uniquely identifies the stairstep. The vertical piece between adjacent stairsteps is longer than $2(k-1)d$.

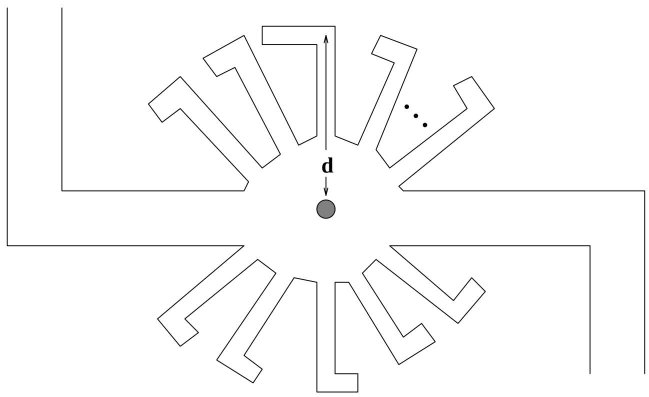

Figure 10: Part of the map polygon that shows Strategy MDL is best possible.

For this map polygon, if the robot is initially placed in the center of the circle on one stairstep, Strategy MDL will direct it to travel up the k protrusions of length d until it finds one that has a longer piece at the end, or until it has examined all but one of these protrusions. In the worst case the robot will travel a distance $2(k-1)d$. Consider any other localizing decision tree strategy. If it directs the robot to travel to any stairstep besides the one where it starts, then the localizing decision tree that it builds will have weighted height greater than $2(k-1)d$. The only way to localize the robot by staying on the stairstep is to direct it to examine the protrusions, and in the worst case the robot must travel a distance $2(k-1)d$ before it has localized itself (assuming that it must return to the origin at the end).

Since no localizing decision tree strategy can build a tree with weighted height less than $k-1$ times the length of an optimal verification tour for all map polygons, Strategy MDL is the best possible strategy.

Note that although Strategy MDL builds a localizing decision tree of weighted height $\Omega(k)$ times the length of an optimal verification tour for the previous two examples, and k may be $\Theta(n)$, these examples *do not* show a lower bound of $\Omega(n)$ for Strategy MDL since each of these map polygons has $\Omega(k^2)$ vertices.

5.4 Creating a reduced set of reference points

The set Q of reference points has size upper bounded by k times the number of cells in arrangement A, which may be very large. In this subsection, we show that when Strategy MDL is run with only a small subset $Q' \subseteq Q$ of the original reference points, the $(k-1)d$ performance guarantee of Section 5.2 still holds. The size of Q' will be no more than $k(k-1)$.

Set Q' is defined as the union of subsets $Q_i \subseteq Q$, where there is one Q_i for each $p_i \in H$ and $|Q_i| \leq k-1$. Ignoring implementation issues, we define Q_i as follows. Initially Q_i is empty, and the subset of Q consisting of reference points $q_{C,i}$ generated for translate P_i is processed in order of increasing $d_i(q_{C,i})$; that is, in order of increasing distance from the origin. For each successive reference point $q_{C,i}$, the partition of H induced by $Q_i \cup \{q_{C,i}\}$ is compared to that induced by Q_i alone. If the subset of H containing location p_i is further subdivided by the additional reference point $q_{C,i}$, then $q_{C,i}$ is added to Q_i. Conceptually, the reference point $q_{C,i}$ is added if it distinguishes another hypothetical initial location from p_i. This process continues until p_i is contained in a singleton in the partition of H induced by Q_i. Since there are only $k-1$ initial locations to be distinguished from p_i, Q_i will contain at most $k-1$ points.

We denote by *Strategy MDLR*, which stands for *Minimum Distance Localization with Reduced reference point set*, the strategy obtained by replacing set Q with Q' in Strategy MDL.

Theorem 6 *Strategy MDLR, which uses a set of at most $k(k-1)$ reference points, localizes the robot by directing it along a path whose length is at most $(k-1)d$, where $k = |H|$ and d is the length of an optimal verification tour for the robot's initial position.*

Proof: Both the proof that Strategy MDLR directs the robot along a path that determines its initial location and the proof of the $(k-1)d$ bound are essentially the same as the proofs of the corresponding results in Theorems 3 and 4 of Section 5.2. The only additional observation needed is that if a reference point $q_{C,i}$ is used in one of the previous proofs to distinguish two hypothetical initial locations, and if $q_{C,i}$ does not belong to set Q', then Q' contains some reference point $q_{C',j}$ that distinguishes the same pair of locations and that satisfies $d_j(q_{C',j}) \leq d_i(q_{C,i})$. Hence, set Q' always contains an adequate substitute for any reference point of Q required by the proofs of Theorems 3 and 4. $\square$

6 Conclusions and Future Research

We have shown that the problem of localizing a robot in a known environment by traveling a minimum distance is NP-hard, and we have given an approximation strategy that achieves a bound of $k-1$ times an optimal solution, where k is the number of possible initial locations of the robot. We have also shown that this bound is the best possible.

The work in this paper is one part of a strategy for localizing a robot. The complete strategy will preprocess the map polygon and store the decision trees for ambiguous initial positions so that the robot only needs to follow a predetermined path to localize itself.

There are many variations to this problem which can be considered. If the robot must localize itself in an environment with obstacles, then the map of the environment can be represented as a simple polygon with holes. In this paper we assigned a cost of zero for the robot to take a probe and analyze it. In a more general setting we would look for an optimal decision tree, where the edges of a decision tree associated with the outcome of a probe would be weighted with the cost to analyze that probe. A pragmatic variation of the problem would weight reference locations so that those that produce more reliable percepts would be selected first.

References

[AI90] David Avis and Hiroshi Imai. Locating a Robot with Angle Measurements. *Journal of Symbolic Computation*, 10:311–326, 1990.

[AMM+93] Esther M. Arkin, Hank Meijer, Joseph S.B. Mitchell, David Rappaport, and Steven S. Skiena. Decision Trees for Geometric Models. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, pages 369–378, San Diego, CA, May 19-21 1993. ACM Press.

[BD90] Kenneth Basye and Thomas Dean. Map learning with indistinguishable locations. In M. Henrion L. N. Kanal J. F. Lemmer, editor, *Uncertainty in Artificial Intelligence 5*, pages 331–340. Elsevier Science Publishers, 1990.

[BG94] Margrit Betke and Leonid Gurvits. Mobile Robot Localization Using Landmarks. Extended abstract, 1994.

[BLM92] Prosenjit K. Bose, Anna Lubiw, and J. Ian Munro. Efficient Visibility Queries in Simple Polygons. In Cao An Wang, editor, *Proceedings of the Fourth Canadian Conference on Computational Geometry*, pages 23–28, St. John's, Newfoundland, Canada, August 10-14 1992. Memorial University of Newfoundland. Also P. Bose, Master's thesis, University of Waterloo, Dec. 1991.

[Bos91] Prosenjit K. Bose. Visibility in Simple Polygons. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, December 1991.

[BR93] Ronen Basri and Ehud Rivlin. Homing using combinations of model views. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pages 1656–1591, Chambery, France, August 1993. Morgan Kaufman Publishers.

[BRS91] Avrim Blum, Prabhakar Raghavan, and Baruch Schieber. Navigating in Unfamiliar Geometric Terrain. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, pages 494–504, 1991. To appear in SICOMP.

[BS91] Patrice Belleville and Tom C. Shermer. Probing Polygons Minimally is Hard. Technical Report CMPT TR 91-12, Simon Fraser University, Burnaby, B.C., Canada, December 1991.

[Cha90] Bernard Chazelle. Triangulating a Simple Polygon in Linear Time. In *Proceedings of the* 31^{st} *Annual Foundations of Computer Science*, pages 220–229. IEEE Computer Society Press, 1990.

[Dav86] Ernest Davis. *Representing and Acquiring Geographic Knowledge.* Pitman and Morgan Kaufmann Publishers, Inc., London and Los Altos, CA, 1986.

[DI94] Amitava Datta and Christian Icking. Competitive Searching in a Generalized Street. In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 175–182, Stony Brook, NY, June 6–8 1994. ACM Press.

[DJMW93] Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. Map validation and self-location in a graph-like world. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence*, pages 1648–1653, Chambery, France, August 1993. Internation Joint Conf. on Artificial Intelligence Inc.

[ELW93] Peter Eades, X. Lin, and N.C. Wormald. Performance Guarantees for Motion Planning with Temporal Uncertainty. *Australian Computing Journal*, 25(1):21–28, February 1993.

[GA81] Hossam A. El Gindy and David Avis. A Linear Algorithm for Computing the Visibility Polygon from a Point. *Journal of Algorithms*, 2:186–197, 1981.

[GHL+87] Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-Time Algorithms for Visibility and Shortest Path Problems Inside Triangulated Simple Polygons. *Algorithmica*, 2:209–233, 1987.

[GMR92] Leonidas J. Guibas, Rajeev Motwani, and Prabhakar Raghavan. The Robot Localization Problem in Two Dimensions. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 259–268, Orlando, FL, January 27–29 1992.

[HR76] Laurent Hyafil and Ronald L. Rivest. Constructing Optimal Binary Decision Trees is NP-Complete. *Information Processing Letters*, 5(1):15–17, May 1976.

[KB87] Benjamin J. Kuipers and Y. T. Byun. A qualitative approach to robot exploration and map-learning. In *Proceedings of the IEEE Workshop on Spatial Reasoning and Multi-Sensor Fusion*, pages 390–404, Los Altos, CA, 1987. IEEE.

[Kle92] Rolf Klein. Walking an Unknown Street with Bounded Detour. *Computational Geometry: Theory and Applications*, 1:325–351, 1992.

[Kle94a] Jon Kleinberg. On-line Search in a Simple Polygon. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 8–15, 1994.

[Kle94b] Jon Kleinberg. The Localization Problem for Mobile Robots. In *Proceedings of the* 35^{th} *Annual Foundations of Computer Science*, pages 521–533, Santa Fe, NM, November 20–22 1994. IEEE Computer Society Press.

[KMK93] Akio Kosaka, Min Meng, and A. C. Kak. Vision-guided mobile robot navigation using retroactive updating of position uncertainty. In *Proceedings of the International Conference of Robotics and Automation, Volume 2*, pages 1–7, Atlanta, GA, May 1993. IEEE Computer Society Press.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Norwell, MA, 1991.

[LP84] D.T. Lee and Franco P. Preparata. Euclidean Shortest Paths in the Presence of Rectilinear Barriers. *Networks*, 14(3):393–410, 1984.

[PY91] Christos Papadimitriou and Mihalis Yannakakis. Shortest Paths without a Map. *Theoretical Computer Science*, 84:127–150, 1991.

[ST85] Daniel D. Sleator and Robert E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28:202–208, 1985.

[TA92] Raj Talluri and J. K. Aggarwal. Position estimation for an autonomous mobile robot in an outdoor environment. *IEEE Trans. on Robotics and Automation*, 8(5):573–584, Oct. 1992.