Roadmap-A^*: an Algorithm for Minimizing Travel Effort in Sensor Based Mobile Robot Navigation

Leon Shmoulian and Elon Rimon*
Technion, Israel Institute of Technology

Abstract

This paper presents an algorithm for minimizing the travel effort of a mobile robot navigating in an unknown environment along a graph constructed on-line by the robot. The graph, called a roadmap*, represents the free configuration-space of the robot, and the travel effort is measured by the length of the path traveled by the robot during the search. The algorithm, called Roadmap-A^*, strives to minimize the travel effort by combining features of the classical A^* and local graph-search algorithms. A user-specified parameter ϵ determines the relative emphasize on the two approaches, where $\epsilon = 0$ corresponds to A^* while $\epsilon = \infty$ corresponds to local-search. We study the performance of the algorithm on roadmaps generated by a cylindrical mobile robot navigating in an unknown environment, using a potential-field roadmap construction method. However, Roadmap-A^* is general and can be used by any on-line roadmap construction method. We study the performance of Roadmap-A^* for various values of ϵ on simulated environments, and indicate how to best choose ϵ for a given class of environments.*

1 Introduction

In sensor-based mobile robot navigation, a robot has to navigate towards a given goal configuration while avoiding collision with obstacles. The obstacles are assumed stationary, but their location is unknown to the robot who must detect their presence using sensors. Some approaches to sensor-based navigation are the works of Chatila [4], Foux [7], Lumelsky [11], Noborio [13], Shiller [22], Stentz [23], and their coworkers. One particular approach incrementally constructs a graph which represents the connectivity of the robot's free configuration-space, called a *roadmap*. The roadmap graph contains the initial and goal configurations, and the navigation problem is reduced to a physical search for the goal along the incrementally constructed roadmap graph. Roadmaps are thus graphs embedded in Euclidean space, such that the cost of traversing an edge is given

*Supported by a grant from the Israel Defense Ministry.

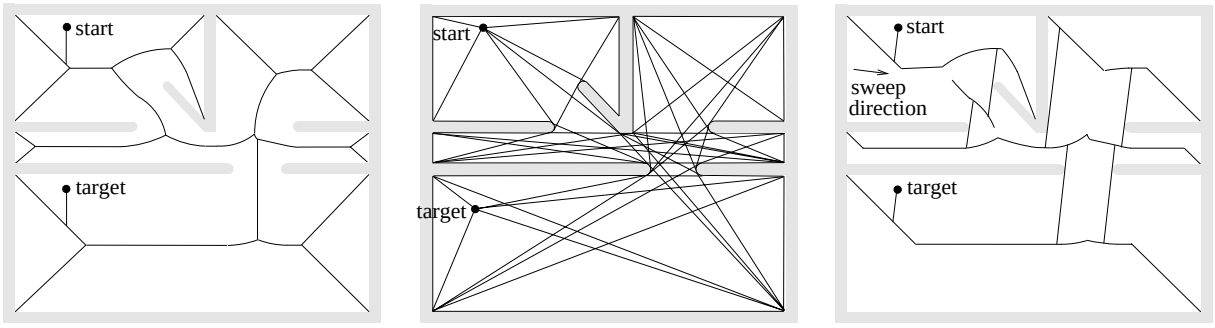


Figure 1: Examples of roadmaps. (a) Voronoi diagram: locus of points equidistant from obstacles. (b) Visibility graph: visibility edges connecting obstacles. (c) Potential-field roadmaps: local maxima of distance from obstacles along pre-determined sweep direction.

by its length. We note that during the search the robot constructs only a portion of the roadmap, not the entire roadmap as required in map-building tasks [17, 24]. Examples of sensor-based roadmaps are: generalized Voronoi diagrams [6], incremental visibility graphs [7], and potential-field based roadmaps [18]. These roadmaps are illustrated for a point robot in the plane in Figure 1.

The sensor-based roadmap methods typically focus on the construction of a connected network of curves. This is certainly the most important property any roadmap must have, as it guarantees that the robot would find a path to the goal if such a path exists. However, these methods pay little attention to the efficiency of the on-line search along the roadmap, which is the focus of this paper. If a robot has perfect apriori information about the environment, it can pre-compute the entire roadmap and perform the search in the computer memory using classical graph-search algorithms such as A^* [8, 15]. But the classical graph-search algorithms are not suitable for robots which have to construct the roadmap incrementally, by *physically* traveling along the roadmap arcs. For instance, A^* requires frequent discontinuous "jumps" during the search, and a robot which attempts to execute these jumps by traveling along the roadmap would perform very poorly. Thus there is a need for a roadmap search algorithm which attempts to reduce the *travel effort*, measured by the length of the path traveled by the robot.

The existing roadmap search algorithms can be divided into two classes. The first class is depth-first search (or DFS) algorithms. In DFS the search agent moves to one of the unvisited neighbors of the current node according to some selection rule. If a deadend is encountered, DFS returns to the closest node along the search path which has unvisited neighbors. Thus DFS is suitable for searching roadmaps, as it avoids undesirable jumps during the search process. An example is the D-DFS algorithm of Rosenfeld *et al.* [19], which takes advantage of the fact that the robot knows the direction to the goal. Another DFS algorithm is $Bug2$ of Lumelsky and Stepanov [12], for navigating a point robot in two-dimensions. (Note, we consider roadmaps in Euclidean space of arbitrary dimension.) $Bug2$ searches along a roadmap consisting of the obstacles' bounding curves, and segments of the line passing through the start and target points. The length of the path traveled by

the robot under $Bug2$ can be bounded as follows [12]. Let S and T be the start and target points. Then the path length l is bounded by $l \leq \frac{1}{2} \sum_i p_i \Pi_i$, where Π_i is the perimeter of the i^{th} obstacle, p_i is the number of intersections of the S-T line with the i^{th} obstacle, and the summation is over the obstacles which intersect the disc with center at T and radius $\|S-T\|$. This bound indicates a deficiency from which all DFS algorithms suffer: in certain environments the path length (and hence the travel effort) is bounded only by the size of the search space.

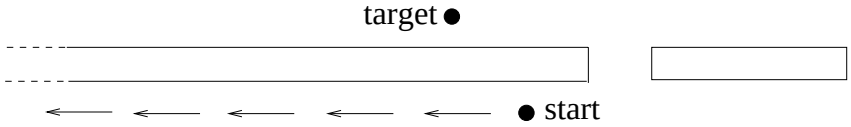


Figure 2: Using *Local-A**, a search agent may drift arbitrarily far from the target. (See Figure 9 for *Roadmap-A** performance in a similar case.)

The other approach taken by roadmap search algorithms is to repeatedly run A^* from the current robot location, an approach we call *Local-A**. In *Local-A** the search agent is at a current node x, and the merit of a node n is determined by the function $f(n) = g(x,n) + h(n,T)$, where $g(x,n)$ is the length of the shortest path from x to n along the known graph, and $h(n,T)$ is a path-length estimate from n to the target[1]. The search agent next moves to the node with minimal f-value, exploring its unvisited neighbors and adding them to the known graph. The search agent thus selects nodes which are close to its current location along the known graph, giving preference to nodes which seem closer to the target. The *Local-A** approach first appeared in Korf's *Real-Time-A** (or *RTA**) algorithm [10] for on-line search of general graphs. We use the term *Local-A** in order to emphasize the local nature of this search approach. *Local-A** is used for roadmap search is the work of Foux *et al.* [7] on searching incremental visibility graphs. Much like DFS algorithms, *Local-A** algorithms do not possess a mechanism for bounding the search depth. This deficiency is illustrated in Figure 2, which shows a long wall separating the start and target. Using *Local-A**, the search agent will be lead away from the target, potentially forever if the wall is infinitely long. The travel effort of *Local-A** can thus become very large, and is again bounded only by the size of the search space. In contrast, the roadmap search algorithm we propose, *Roadmap-A**, imposes an output-sensitive bound on the search depth in terms of the length of the shortest path to the target.

The *Roadmap-A** algorithm executes a series of *Local-A** probes, whose depth is bounded by a higher-level A^*_ϵ algorithm. The A^*_ϵ algorithm is a variant of the classical A^* [16], in which the search agent may choose to expand any node from a set called focal. This set consists of open nodes whose value (as determined by the classical A^*) is ϵ-away from the best open-node value. As such, focal introduces an additional degree of freedom into A^*, which allows *Roadmap-A** to apply minimum-travel-effort considerations in the selection of nodes for expansion. The A^*_ϵ algorithm has properties similar to those of A^*. In particular, A^*_ϵ imposes a bound on the search depth which depends on the length of the

[1]This approach can also be viewed as best-first-search with a local cost function that depends on x.

shortest path, and it always finds an ϵ-optimal path to the target. *Roadmap*-A^* inherits these important from the A^*_ϵ algorithm. Another important feature of *Roadmap*-A^* is the way it handles node expansion. Usually expansion requires exploration of every unvisited neighbor of the current node. But for a mobile robot such an expansion would mean *costly excursions* to all unvisited neighbors of the current node. *Roadmap*-A^* breaks the node expansion into a series of single-arc explorations using a node-splitting technique. This technique reduces the travel effort during node expansion, yet keeps the operation of *Roadmap*-A^* within the framework of A^*_ϵ.

In the related literature, another algorithm by Korf called IDA^* shares common features with *Roadmap*-A^*. IDA^* is designed for off-line search of exponential trees, which are typically too large to be stored in the computer memory. Korf describes IDA^* as follows [9]. "IDA^* is a modification of A^* that reduces its space complexity from exponential to linear. IDA^* performs a series of depth-first searches, in which a branch is cut off when the cost of its frontier node, $f(n) = g(n) + h(n)$, exceeds a cutoff threshold. The threshold starts at the heuristic estimate of the initial state, and increases at each iteration to the minimum value that exceeds the previous threshold, until the solution is found. IDA^* has the same property as A^* with respect to solution optimality, and expands the same number of nodes, asymptotically, as A^* on an exponential tree, but uses only linear space." The IDA^* and *Roadmap*-A^* algorithms are similar in the sense that both employ local search (either DFS or *Local*-A^*), which is dynamically bounded by a higher-level A^* algorithm. However, the two algorithms strive to satisfy different objectives. While IDA^* is an off-line algorithm that strives to emulate A^* in linear space, *Roadmap*-A^* is an on-line algorithm that strives to reduce physical travel effort.

We test the proposed algorithm on roadmaps generated by a potential-field based method called *IRoadmap* [18]. This approach has been previously described as a general paradigm, and we provide here the details necessary for its implementation on a cylindrical mobile robot equipped with range sensors. Although we test *Roadmap*-A^* on roadmaps generated by a particular method, the algorithm can be used by any on-line roadmap construction method. *Roadmap*-A^* has the following additional characteristics. First, the algorithm can run on roadmaps of arbitrary size, as long as the roadmap can be stored in the computer memory. Second, the algorithm can in principal run on roadmaps representing the free configuration-space of general mechanisms, although here we emphasize the two-degree-of-freedom navigation problem of a cylindrical mobile robot. Third, the algorithm can run on any graph constructed on-line by a search agent, not necessarily on graphs embedded in Euclidean space. In the latter case all is required is an *admissible* cost estimate $h(n, T)$, and a *monotonic* cost function $f(n)$. These terms are reviewed below, but all reasonable cost functions do have these two properties [10].

The paper is organized as follows. We begin with a description of *Roadmap*-A^*. Then we analyze some properties of *Roadmap*-A^*, showing that monotonicity of the cost function allows an efficient implementation of the algorithm. Next we review the sensor-based roadmap construction method *IRoadmap*, which is used in the experimental studies of the algorithm. The travel effort of *Roadmap*-A^* is strongly influenced by the user-specified parameter ϵ. The value of $\epsilon = 0$ corresponds to the classical A^* algorithm and

$\epsilon = \infty$ corresponds to pure local-search behavior. In Section 5 we experimentally study the dependency of the travel effort on ϵ, showing that each class of environments has an optimal range of ϵ-values. In the concluding section we summarize the contributions of this work and discuss several relevant issues.

2 The *Roadmap-A** Algorithm

We consider a physical search agent (a robot in our case), which has to search for a target node on an incrementally constructed graph G. The graph is embedded in Euclidean space and the target coordinates are known to the search agent. First we define a few terms used by the algorithm, then we present the algorithm itself.

2.1 Basic Terminology

The following terminology is used in the description of the algorithm and its subsequent analysis. When a node has been visited by the search agent, it is said to be *generated*. Note that the search agent has no knowledge of the neighbors of a newly generated node, until it physically travels along the arcs leading to these neighbors. When an arc has been traced by the search agent, it is said to be *explored*. If all the arcs of a node are explored, the node is said to be *expanded*. Node expansion requires that the agent physically trace every arc emanating from the node, a process which may consume considerable travel effort. To facilitate this node expansion process, we introduce the following operation of *node splitting*. Let n be a node with k arcs, as depicted in Figure 3. Then n is split into k *subnodes* interconnected by zero-length arcs, and the subnodes become regular nodes in the graph. The splitting operation increases the number of nodes by a factor equal to the branching factor. However, now each subnode has only one non-zero arc emanating from it, and the expansion of a subnode requires exploration of a *single arc*. Moreover, the zero-length arcs introduced by the splitting operation do not affect the algorithm's convergence, since at each step the agent must explore one non-zero arc.

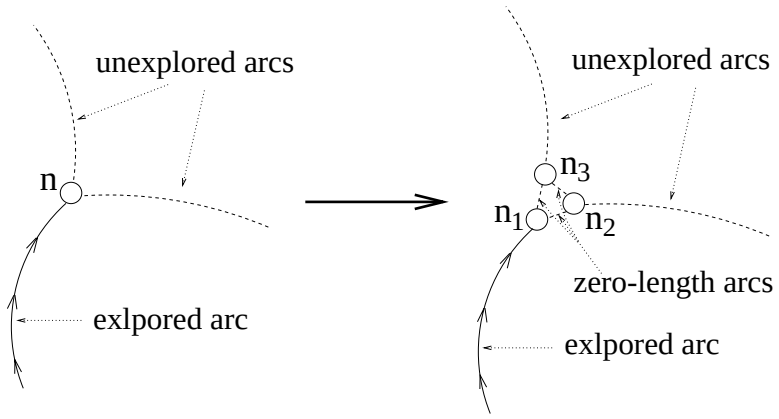


Figure 3: The splitting of a node having three arcs into three subnodes.

As the search agent incrementally constructs the graph G, it has knowledge only of the currently constructed subgraph, denoted G'. We now introduce several terms associated with G'. The start node is denoted S; the node that the search agent is currently visiting is denoted x; and the target node, which becomes a node of G' only at the end of the search, is denoted T. The cost of an arc in G' is denoted $c(n, n')$, where n and n' are the nodes at the endpoints of the arc. (The splitting operation ensures that there is no ambiguity with this notation.) Note that the cost of an arc is known only after the search agent has explored the arc and added it to G'. A sequence of nodes $n_1, n_2, ..., n_k$, where n_{i+1} is a neighbor of n_i in G' is called a *path*. The cost of a path is measured by the sum of the costs of the arcs along the path. Finally, two nodes n_i and n_j may be connected in the subgraph G' via several paths, and $c(n_i, n_j)$ denotes the minimal cost of these paths.

2.2 The Selection Functions of *Roadmap-A**

We introduce three selection functions which organize the execution of *Roadmap-A**. *Roadmap-A** uses a hierarchy of two algorithms to combine local search with global travel-effort considerations. The higher-level algorithm is A^*_ϵ (reviewed in the appendix), which allows the search agent to choose for expansion any node from a set called focal. *Roadmap-A** uses the first selection function to determine the focal set. The lower-level algorithm is *Local-A**, which selects from focal a node which tends to minimize the travel effort. The selection of a node from focal by *Local-A** is governed by the other two selection functions. A description of the selection functions follows.

Recall that $h(n, T)$ estimates the length of the shortest path from a node n to the target along the graph G. In roadmap search $h(n, T)$ is usually chosen as the Euclidean or Manhattan distance from n to T, but any *admissible*[2] function h can be used for this purpose. The first function used by *Roadmap-A** is a global cost function, $f_{glob}(n)$, that estimates the length of the shortest path from the start to the target through the node n. Since only the subgraph G' is known, the shortest path from S to n is computed along G', and its length is denoted $c(S, n)$. The cost of the path from n to T is estimated as $h(n, T)$, and the function $f_{glob}(n)$ is given by $f_{glob}(n) = c(S, n) + h(n, T)$. The algorithm uses $f_{glob}(n)$ to determine the nodes of focal as follows:

$$\text{focal} = \{n \in \text{open} : f_{glob}(n) \leq (1 + \epsilon) f_{glob}(n_0)\},$$

where open is the set of nodes which have been generated but not yet expanded, n_0 is the node with minimum f_{glob}-value in open, and ϵ is a user-specified parameter. The inclusion of S in the definition of $f_{glob}(n)$ has the effect of focusing the search around the shortest path from S to T, as explained in the discussion following Corollary 3.2. The focal set introduces a degree of freedom which allows the algorithm to select nodes for expansion based on minimum-travel-effort considerations.

The second function used by *Roadmap-A** is a local cost function, that estimates the length of the shortest path from the *current* node to the target through a node n, which

[2] $h(n,T)$ is admissible if it is non-negative and never overestimates the true cost of the path from n to T.

is already in the known graph G'. The local cost function is denoted $f_{loc}(n)$ and is given by $f_{loc}(n) = c(x,n) + h(n,T)$, where $c(x,n)$ is the length of the shortest path from the current node x to n along G'. The algorithm uses $f_{loc}(n)$ to estimate which node of focal involves the least travel effort and should therefore be selected next for expansion. The function $f_{loc}(n)$ attains its minimum value at the subnodes associated with the physical node currently visited by the search agent. This property endows the algorithm with a local-search behavior, where the search agent prefers to explore nodes which are close to its current location along the known graph.

The third function used by *Roadmap-A** is a tie-breaking function. By construction, subnodes associated with the same physical node have the same local cost $f_{loc}(n)$. The tie-breaking function is denoted $f_{dir}(n)$ and is defined as follows. Let e_{arc} be a unit vector tangent to the non-zero arc emanating from a subnode n, and let e_{goal} be a unit vector in the direction from n to T. Then $f_{dir}(n)$ is given by the scalar product $f_{dir}(n) = e_{arc} \cdot e_{goal}$. When confronted with a choice among subnodes having the same local cost, the algorithm selects the subnode with the largest f_{dir}-value, as the direction of the arc emanating from this subnode is the closest to the direction to the target. More sophisticated tie-breaking functions can be defined, but the principal operation of *Roadmap-A** does not depend on the particular tie-breaking function being used.

2.3 Description of *Roadmap-A**

We now give a schematic description of the algorithm, followed by a description of two sub-algorithms used by *Roadmap-A**.

Input: A value for the sub-optimality parameter ϵ.

***Roadmap-A** Algorithm:**

1. Split the start node S into subnodes. Put the subnodes in open and focal.
2. If open is empty, exit with **failure**.
3. Run *Local-A** (see below) on the known graph G', to find a node n in focal with minimal local cost $f_{loc}(n)$. If several nodes have the same f_{loc}-value, select the node with maximal directional cost $f_{dir}(n)$. Remove n from open and focal, and put it in close. **Execute** a move to n along the known graph G'.
4. If n is the target node T, exit with the **solution**.
5. Otherwise, *expand* the node n by **moving** along its non-zero arc and generating its successor node n'. Then execute:
   (a) If n' is neither in open nor in close, it is a new node. If n' has no other arcs emanating from it, prune n' from G' and return to n. Otherwise, *split* n' into subnodes and add the subnodes to open with a pointer to n. Go to step 6.
   (b) Compute the global cost of n', which is already in open or close. If the new $f_{glob}(n')$ is less than the old one, update $f_{glob}(n')$ and change the pointer of n' to n. Update the nodes in G' using *cost propagation* (see below).
6. Update focal using the rule: $\text{focal} = \{n \in \text{open} : f_{glob}(n) \leq (1+\epsilon) f_{glob}(n_0)\}$ where $f_{glob}(n_0) = \min_{m \in \text{open}} \{f_{glob}(m)\}$. Go to step 2.

Before describing the two sub-algorithms used by *Roadmap*-A^*, let us clarify some issues concerning the algorithm. First, in step 3 the node n is actually a subnode, and step 4 should be more accurately phrased as arrival to any subnode associated with the target node. Second, in step 5(a) the successor node n' is a new node which has not been split yet, while in step 5(b) the successor node n' is already a subnode. Third, in roadmap search the search effort is strongly dominated by the time it takes the search agent to physically travel along the roadmap arcs. Other search-effort components, such as computation time, are relatively negligible in this type of a problem. Thus, in step 3 the algorithm runs a *Local*-A^* probe along the known graph in the computer memory, and this step takes a relatively negligible time. In contrast, the move to the next node for expansion at the end of step 3, and the tracing of an arc during node expansion in step 5, involve a physical travel and constitute the main search effort. We now describe the two sub-algorithms used by *Roadmap*-A^*.

Local-A^*: *Roadmap*-A^* chooses for expansion the node n with minimal f_{loc}-value in focal. The function $f_{loc}(n)$ is given by $f_{loc}(n) = c(x,n) + h(n,T)$, where x is the search agent current node. To find the node with minimal f_{loc}-value in focal, the search agent runs a *Local*-A^* along the known graph G', starting at the current node x and using f_{loc} for node evaluation. However, while the node T serves as a nominal target, the search terminates once a node from focal is chosen for expansion. As discussed below, f_{loc} is a monotonic cost function, and consequently the local cost of the nodes chosen for expansion increases monotonically. Moreover, the cost of a node does not change once it has been expanded. It follows that the first node chosen for expansion by *Local*-A^* from focal is the node with minimal f_{loc}-value in focal, as required. The *Local*-A^* search also generates the shortest path from x to the node with minimal f_{loc}-value, and the search agent moves along this path to the new node. Furthermore, if any subnode associated with the current node x is in focal, this node would automatically have the minimal f_{loc}-value and be chosen for expansion. Otherwise, one of the immediate neighbors of x would be chosen for expansion by the *Local*-A^* sub-algorithm. This property of *Local*-A^* endows the *Roadmap*-A^* algorithm with the desired local-search behavior, where the search agent prefers to expand nodes in its immediate vicinity.

Cost propagation: When a node n is expanded, the search agent traces a non-zero arc until it reaches a node n'. If the path to n' through the arc is shorter than the current path to n', the algorithm updates the global cost of n' and changes the pointer of n' to n. Next the algorithm recursively propagates this change to the other nodes in G' as follows[3]. At the basis of the recursion comes the updating of n' itself. Now given a node m which has just been updated, let m' be a neighbor of m in G'. The algorithm computes the global cost of m' as $f_{glob}(m') = c(S,m) + c(m,m') + h(m',T)$, where $c(S,m)$ is the length of the new path from S to m through n, and $c(m,m')$ is the cost of the arc from m to m'. If the new global cost of m' is less than the old one, the algorithm updates the global cost of m' and changes the pointer of m' to m. As discussed below, f_{glob} is also a monotonic

[3]Unlike the classical A^*, the use of focal makes it possible for the global cost of an expanded node to decrease at a later stage. Hence the cost propagation includes all the nodes in G'.

cost function, and this property ensures that the recursion ends in a finite number of steps. The cost-propagation process always decreases the global cost of nodes in G', and as a result more nodes of G' move into focal. A larger focal set means a larger selection available for the *Local-A** sub-algorithm, resulting in an enhanced travel-effort saving at the local-search level. We note that there exists an alternative technique for updating the cost of nodes, called *node reopening* [15, p. 49]. However, the latter technique does not strive to put as many nodes in focal as possible, resulting in a diminished travel-effort saving at the local-search level.

Finally we mention a possible modification to the algorithm that can further reduce the travel effort. In the definition of the algorithm, the agent can explore a non-zero arc twice. This can happen in the following two ways. The first is when the agent reaches a new physical node n. The agent enters the node through a non-zero arc, and one of the subnodes of n becomes associated with this arc. The second is when the agent completes a loop and returns to a previously visited physical node m. It enters the node through a non-zero arc which is associated with a subnode of m. In both cases let us call the subnode associated with the entry arc an *entry subnode*. The entry subnodes are initially in open, although their non-zero arc has been already explored by the agent. Currently, when an entry subnode is chosen for expansion, the search agent moves to this subnode and traces its non-zero arc for the second time. We can avoid this redundant move by marking the entry subnodes as candidates for instant closing. Once an entry subnode joins focal, it can be instantly placed in close, since its non-zero arc has already been explored. This closing avoids unnecessary moves to entry subnodes, and allows the search agent to immediately proceed with the expansion of other nodes from focal.

3 Analysis of *Roadmap-A**

In this section we discuss several properties of *Roadmap-A**. To begin with, *Roadmap-A** belongs to the A^*_ϵ family of algorithms. As such, it always terminates on finite graphs and finds a path to the target whenever such a path exists [15, Sec. 3.1]. The following two corollaries are important to the understanding of the algorithm. The first corollary asserts that *Roadmap-A** explores a portion of the graph that contains an ϵ-optimal solution.

Corollary 3.1 *Upon reaching the target, the portion of the graph explored by Roadmap-A* contains a path from S to T whose length l satisfies $l \leq (1+\epsilon)l_{opt}$, where l_{opt} is the length of the shortest path from S to T in G.*

The corollary follows from a similar property of A^*_ϵ, see Theorem 1 in the appendix. The theorem in the appendix requires that the function $h(n,T)$ be admissible, which holds true for the Euclidean distance used by *Roadmap-A**. The corollary provides assurance that *Roadmap-A** explores a portion of the graph G which is most relevant for navigating from S to T. The next corollary asserts that the algorithm expands only nodes which lie

in an ellipse whose size[4] is determined by the parameter ϵ and the length of the shortest path from S to T.

Corollary 3.2 *When Roadmap-A^* selects a node n for expansion, the node satisfies*

$$\|n - S\| + \|n - T\| \leq (1 + \epsilon) l_{opt}, \tag{1}$$

where l_{opt} is the length of the shortest path from S to T in G. Thus Roadmap-A^* expands only those nodes which lie in an ellipse with focal points S and T and size $(1 + \epsilon) l_{opt}$.

The corollary follows from a similar ellipse property of A^*_ϵ stated in the appendix. In general the roadmap graph is embedded in a Euclidean space of arbitrary dimension, and the region defined by (1) is more precisely an ellipsoid-of-revolution, obtained by rotating an ellipse with focal points S and T and size $(1+\epsilon)l_{opt}$ around its major axis. However, for convenience we refer to this region as an ellipse. The result stated in the corollary can be sharpened as follows. At any stage of the search process, draw an ellipse with focal points at S and T and size $(1+\epsilon)f_{glob}(n_0)$, where $f_{glob}(n_0)$ is the minimal global cost in open. Then it can be verified that all the currently expanded nodes and all the nodes in focal lie within this ellipse. In particular, when *Roadmap-A^** terminates, the entire collection of expanded nodes and the nodes of focal lie inside the ellipse with focal points S and T and size $(1+\epsilon)l_{opt}$. The practical meaning of the ellipse property is as follows. If l_{opt} is close to the straight-line distance from S to T and ϵ is small, the ellipse is rather "skinny" and not much longer than the Euclidean distance from S to T. In such a case the portion of the roadmap which is searched is rather small. Considering that the roadmap can stretch out far beyond such an ellipse, this is a pleasing output-sensitive result.

The corollary also provides a formal justification for the inclusion of the term $c(S, n)$ in the global cost function, $f_{glob} = c(S, n) + h(n, T)$. To illustrate this point, suppose that the algorithm uses the function $f'_{glob}(n) = h(n, T)$. In that case Corollary 3.2 would apply to f'_{glob}, and all nodes selected for expansion will satisfy $\|n - T\| \leq (1 + \epsilon) l_{opt}$. In other words, the algorithm will expand nodes in the disc centered at T with radius $(1 + \epsilon) l_{opt}$. However, this disc contains the ellipse with focal points at S and T in its interior, since $\|n - S\| + \|n - T\| \leq (1 + \epsilon) l_{opt}$ implies that $\|n - T\| < (1 + \epsilon) l_{opt}$. It follows that the inclusion of S in f_{glob} reduces the space explored by the search agent, and focuses the search to the vicinity of the shortest path from S to T.

However, the ellipse bound is lacking in two important ways. First, the search agent also visits open nodes which are not in focal, and these nodes do not necessarily lie inside the ellipse. (But note that the open nodes lie at most one arc away from the ellipse.) The open nodes can be bounded in a larger ellipse as follows. Let l_{arc} be a maximal arc-length parameter, and let the search agent define intermediate nodes at intervals of length l_{arc} along each arc it travels. The extra nodes would not change the basic operation of the algorithm, but now *all* the nodes visited by the search agent lie within a larger ellipse given by $\|n - S\| + \|n - T\| \leq (1 + \epsilon) l_{opt} + 2 l_{arc}$. Although this technique provides the desired

[4] The size of an ellipse is the sum of the distance of points on the ellipse from the ellipse's focal points.

bound, it is currently not clear how to best choose the parameter l_{arc}. The second caveat is that the ellipse bounds the search space, not the travel effort of the search agent. In practice, roadmaps tend to have evenly distributed nodes. In such roadmaps the average number of nodes inside the ellipse is determined by the size of the ellipse. Denoting this number by #nodes, the travel effort can be bounded in terms of #nodes as follows. In each step of the algorithm, the search agent moves from the current node x to a node n along the shortest path in the known graph G'. According to Theorem 1 in the appendix, the length of the path from S to any node selected for expansion is bounded by $(1+\epsilon)l_{opt}$. Since x and n can be connected by a path through the start node in G', the length of the shortest path from x to n in G' is at most $2(1+\epsilon)l_{opt}$. The total length of the path traveled by the search agent, denoted l, is thus bounded by[5] $l \leq 2(1+\epsilon)l_{opt}$#nodes. An alternative approach would be to measure the average performance of the algorithm in various simulated environments, and this approach is taken in the next section.

In the remainder of this section we discuss properties of the focal set which allow its efficient implementation. First we establish that the global cost function f_{glob} is monotonic. By definition, f_{glob} is *monotonic* if it is non-decreasing along paths leading away from the start node. In other words, if n' is a direct successor of n, then $f_{glob}(n) \leq f_{glob}(n')$. But $f_{glob}(n) = c(S,n) + h(n,T)$ and $f_{glob}(n') = c(S,n) + c(n,n') + h(n',T)$, with $c(n,n')$ being the cost of the arc joining n with n'. Thus the inequality $f_{glob}(n) \leq f_{glob}(n')$ is equivalent to the condition $h(n,T) \leq c(n,n') + h(n',T)$, which is a generalized triangle inequality. A function $h(n,T)$ satisfying the generalized triangle inequality is said to be *consistent*, and we note that consistency implies admissibility. The following lemma asserts that the function $h(n,T) = \|n - T\|$ used by *Roadmap-A*** is consistent.

Lemma 3.3 *The path-length estimate used by Roadmap-A*, $h(n,T) = \|n - T\|$, is consistent. Hence the cost functions f_{glob} and f_{loc} are monotonic.*

Proof: According to the usual triangle inequality, $\|n - T\| \leq \|n - n'\| + \|n' - T\|$. In our case $h(n,T) = \|n - T\|$ and $h(n',T) = \|n' - T\|$. Hence $h(n,T) \leq \|n - n'\| + h(n',T)$. But $c(n,n') \leq \|n - n'\|$, since $c(n,n')$ is the length of the arc joining n with n'. Therefore $h(n,T) \leq c(n,n') + h(n',T)$, and the function $h(n,T) = \|n - T\|$ is consistent. □

The monotonicity of f_{glob} plays a role during cost propagation, as it guarantees that the updating of the global cost of nodes in G' would end in a finite number of steps. The monotonicity of f_{loc} is used in the running of *Local-A*** from the current node, in order to find the node with minimum f_{loc}-value in focal. We need the following lemma, which guarantees that during the search process, *Roadmap-A*** only improves the global cost of every node in the known graph.

Lemma 3.4 *The global cost f_{glob} of each node in the known graph G', expanded or unexpanded, decreases monotonically during the search process.*

[5] The area of the ellipse is given by $\frac{\pi}{4}(1+\epsilon)l_{opt}[(1+\epsilon)^2 l_{opt}^2 - \|S-T\|^2]^{1/2}$. Assuming a uniform density of ρ nodes per unit area, $l \leq \frac{\pi}{2}\rho(1+\epsilon)^2 l_{opt}^2[(1+\epsilon)^2 l_{opt}^2 - \|S-T\|^2]^{1/2}$.

Note that the lemma refers to the cost of a *particular* node in G', not to the cost of nodes along paths leading away from S.

Proof: The global cost of a node n is given by $f_{glob}(n) = c(S,n) + h(n,T)$. Out of the two components of f_{glob}, only the cost of the path $c(S,n)$ along the known graph G' may change, while $h(n,T)$ remains constant. The only way by which $c(S,n)$ can change is when a shorter path from S to n is found. Hence the value of $f_{glob}(n)$ for a particular node n can only decrease during the search process. □

The following proposition asserts that the best node in open remains the same node until it is expanded.

Proposition 3.5 *If at any stage of the search a node has the minimal global cost f_{glob} in* open*, this node remains the best node in* open *until it is selected for expansion.*

Proof: Let n be the node with the minimal f_{glob}-value in open. According to Lemma 3.4, $f_{glob}(n)$ does not increase during the search process. We now show that $f_{glob}(n)$ also does not decrease. Let $m \in$ focal be the node chosen next for expansion. Then by assumption $f_{glob}(m) \geq f_{glob}(n)$. Suppose that after the expansion of m, $f_{glob}(n)$ has decreased to $f'_{glob}(n)$. This change in the global cost of n can occur after a pointer from m to n has been installed, or after cost propagation has been executed (step 5(b) in the algorithm). In both cases we have that n is a successor of m, and the monotonicity of f_{glob} implies that $f_{glob}(m) \leq f'_{glob}(n)$. Hence, if the new value $f'_{glob}(n)$ is smaller than the old one, it must be true that $f_{glob}(m) < f_{glob}(n)$. But this contradicts our initial assumption that $f_{glob}(m) \geq f_{glob}(n)$. Hence $f_{glob}(n)$ cannot decrease.

We also have to show that no other node than n can become the best node in open after the expansion of m. There are two possible cases to consider. A node z which is already in open has become better than n, or a new node z which has been added to open has a lower f_{glob}-value than n. In both cases $f_{glob}(z) < f_{glob}(n)$ after the expansion of m. But in both cases the node z is necessarily a successor of m, which implies by the monotonicity of f_{glob} that $f_{glob}(z) \geq f_{glob}(m)$. This leads to a contradiction, since by assumption $f_{glob}(n) \leq f_{glob}(m)$ and consequently $f_{glob}(z) \geq f_{glob}(m) \geq f_{glob}(n)$. □

The next proposition asserts that the best node in open has a monotonically increasing global cost.

Proposition 3.6 *The minimal global cost of nodes in* open *increases monotonically during the search process.*

Proof: We already know from Proposition 3.5 that the minimal global cost in open changes only when the node with this cost is selected for expansion. Suppose that after an expansion of the best node n, a node n' has the minimal global cost in open. If n' was in open prior to the expansion of n, then either the global cost of n' has not changed, or it has changed and then n' is necessarily a successor of n. In the first case we have that $f_{glob}(n) \leq f_{glob}(n')$ since n was the best node in open, and in the latter case we have

that $f_{glob}(n) \leq f_{glob}(n')$ by the monotonicity of f_{glob}. If n' was not in open, it must have appeared as a result of the expansion of n. This also means that n' is a successor of n, and by the monotonicity of f_{glob}, $f_{glob}(n) \leq f_{glob}(n')$. Thus in all possible cases the global cost of the new best node n' is equal or larger than the global cost of the previous best node n. □

The algorithm must re-compute the focal set at each iteration, and this computation can be efficiently performed as follows. According to Proposition 3.5, the minimum f_{glob}-value in open, $f_{glob}(n_0)$, is piecewise constant. As long as $f_{glob}(n_0)$ remains constant, nodes in open which have been rejected from entering focal will stay out of focal until their global cost decreases due to cost propagation. Moreover, according to Lemma 3.4, the global cost of each particular node decreases monotonically. Hence no node ever leaves focal except when the node is selected for expansion. This result holds true even when $f_{glob}(n_0)$ changes, since $f_{glob}(n_0)$ can only increase according to Proposition 3.6. To summarize, at each iteration the algorithm has to consider only nodes which are candidates for entering focal. At an iteration where $f_{glob}(n_0)$ has not changed, the candidate nodes are either the new nodes in open, or nodes which are already in open whose cost has decreased due to cost propagation. At an infrequent iteration where $f_{glob}(n_0)$ has increased, the candidate nodes are all the nodes in open which are not already in focal. This understanding of the possible changes in focal allows its updating in optimal time i.e., in time proportional to the number of nodes that can possibly join focal.

4 Incremental Roadmap Construction

We describe a roadmap construction method called *IRoadmap*, that is used below for testing the *Roadmap*-A^* algorithm. This method has its origin with the work of Canny and Lin [3], who introduced the approach in the context of *off-line* planning. The *on-line* version of the method is described in [18], but only in general terms applicable to any robot. Here the method is instantiated for a cylindrical mobile robot, and the details required for its implementation using range sensors are given in the appendix. The method has been implemented and tested on a Nomad mobile robot equipped with a sonar ring.

The roadmap is constructed in the robot's configuration space, and we first review this representation. The location of a cylindrical robot is uniquely determined by the coordinates of its center, denoted (r_x, r_y). These coordinates form the configuration space, or *c-space*, of the robot. The configuration space is populated by forbidden regions which represent the physical obstacles as follows. Given an obstacle $\mathcal{B}_i$, its *c-obstacle* $\mathcal{CB}_i$ is the set of configurations at which the robot intersects the obstacle. For a cylindrical mobile robot, $\mathcal{CB}_i$ is obtained by "growing" the obstacle $\mathcal{B}_i$ by a layer of thickness R, where R is the radius of the robot (Figure 5(b)). The complement of the c-obstacles' interiors, called the *freespace* $\mathcal{F}$, is the collection of configurations where the robot may move without hitting obstacles. The planning of the robot motions is consequently performed in $\mathcal{F}$, and the roadmap graph represents the connectivity of this space.

In *IRoadmap* the robot uses three sensors. A position sensor which measures the

robot's current location, a range sensor which measures the minimum distance of the robot to the surrounding obstacles, and a novel sensor which detects *critical points* in the robot's configuration space. The latter sensor is described below. The robot encodes the range data as a repulsive potential field and uses it as illustrated in Figure 4. The robot first determines a *sweep direction* and regards the freespace as a series of slices along this direction. (The specific sweep direction affects the resulting roadmap, but not its connectivity.) The roadmap contains two families of curves. The first family consists of *ridge curves*, which correspond to local-maxima of the repulsive potential associated with the range data. The other family consists of *linking curves*, which pass through critical points and connect neighboring ridge-curves. Next we discuss the construction of these two families of curves.

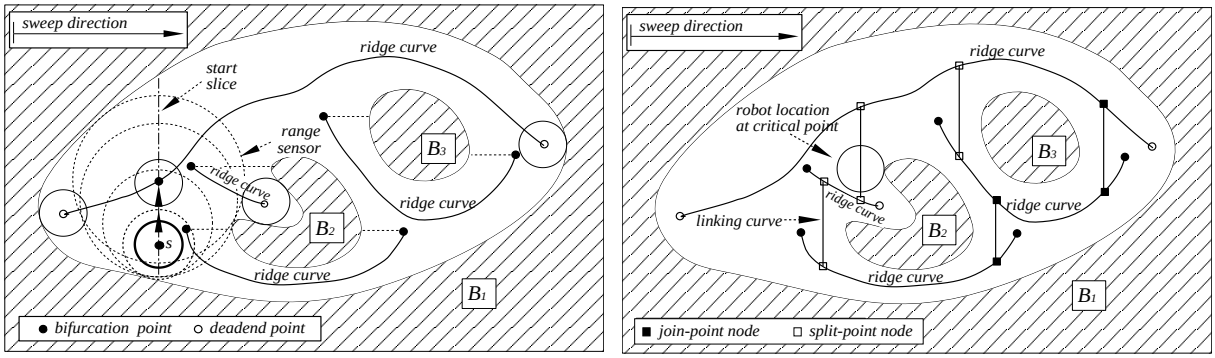
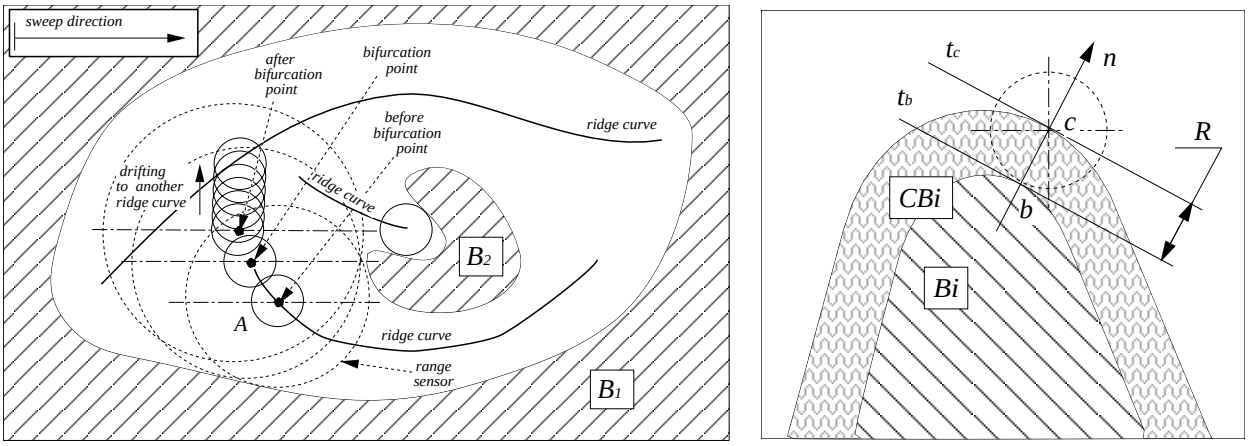


Figure 4: (a) The ridge curves and their endpoints. (The robot is shown at selected locations.) (b) The linking curves through join and split points.

To trace a ridge-curve, the robot initially moves in the slice of the start configuration along increasing value of the repulsive potential, until it reaches a local maximum. This local maximum is a point on the ridge-curve. Then the robot follows the ridge-curve by tracing the local-maxima along the sweep direction, as illustrated in Figure 4(a). The linking-curves pass through critical points where the connectivity of the freespace slices changes. These critical points are either *join points* where two locally distinct components of the freespace slices meet, or *split points* where a connected component splits along the sweep direction. Each linking-curve passes through a particular critical point and connects two neighboring ridge-curves, as shown in Figure 4(b). To construct a linking-curve, the robot first moves in the slice of the critical point until it reaches the critical point, then it continues along increasing value of the repulsive potential until it reaches a neighboring ridge-curve. Eventually the robot traces a ridge-curve whose basin of attraction includes the target configuration, from which the target is directly accessible. It has been shown in Ref. [18] that the resulting collection of curves always forms a connected network which contains a path to the target.

The nodes of the roadmap are of the following two types. The first type are ridge-curve endpoints, which are either deadend points where a ridge-curve reaches the boundary of the freespace, or endpoints in the interior of the freespace. In the latter case the ridge-curve endpoint is a bifurcation point, designating a slice beyond which points are repulsed

to a neighboring ridge curve (Figure 5(a)). When the robot reaches a ridge-curve endpoint, it defines a node at this point and reverses its direction along the ridge curve. The second type of nodes occur when the robot detects a critical point, which causes branching along the linking-curve associated with the critical point. The critical points are detected by a novel "sensor" called the *critical point detector*. This detector is required to locate *all* the critical points (join and split points) in the basin of attraction of the current ridge-curve, and its implementation is discussed in the appendix. The robot defines a node at the point where the linking-curve branches off from the ridge curve, and a second node at the point where the linking-curve reaches the neighboring ridge curve.



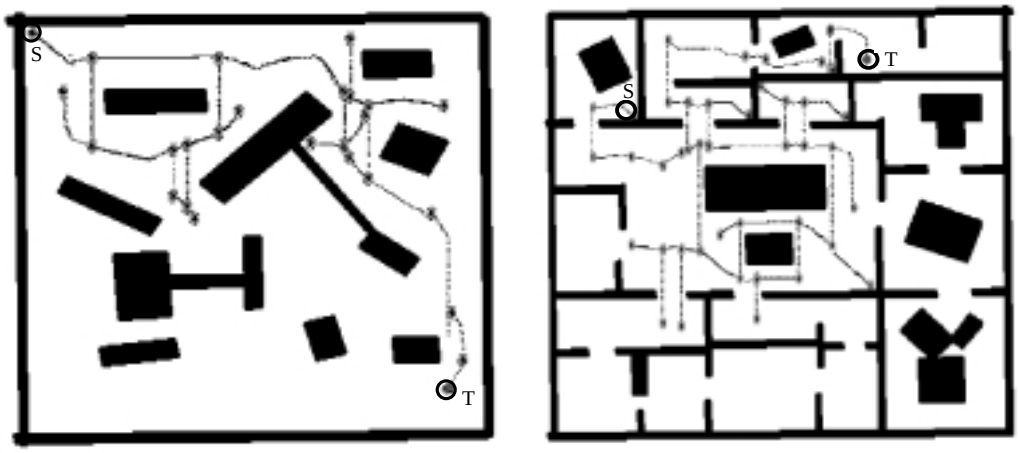
(a) (b)

Figure 5: (a) Detection of a bifurcation point. (b) The tangent to $\mathcal{CB}_i$ at c is parallel to the tangent to $\mathcal{B}_i$ at b.

When a critical point is detected, the robot may choose to continue along the ridge-curve, or to first build a linking-curve through the critical point to a neighboring ridge-curve. But *IRoadmap does not specify which pieces of the roadmap the robot should explore first during the search.* Similarly, other on-line roadmap construction methods do not specify how to efficiently conduct the search along the roadmap. The *Roadmap-A** algorithm we presented provides a framework for making such decisions, in a way which tends to minimize the robot's travel effort.

5 Influence of ϵ on Travel Effort

The choice of the sub-optimality parameter ϵ strongly influences the travel effort of *Roadmap-A**. In order to investigate this influence, we measured the average travel effort as a function of ϵ in two types of simulated environments. The first type, called room environment, is a simple environment consisting of a rectangular room populated by randomly located polygonal objects representing pieces of furniture (Figure 6(a)). The second type, called office floor environment, is a rather complex environment consisting of offices interconnected by doors and corridors and populated by objects representing pieces



(a) (b)

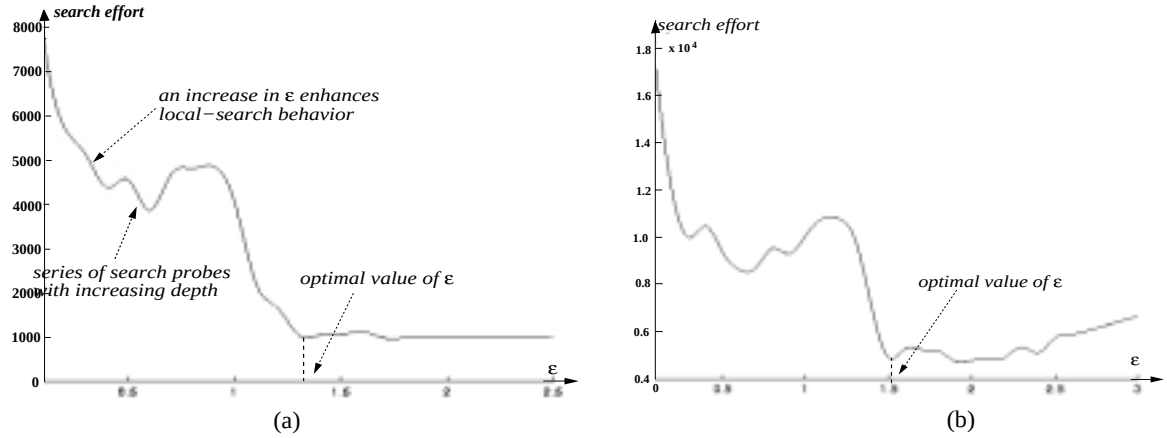
Figure 6: Top view of (a) a room environment, and (b) an office floor environment. The roadmaps are generated by a cylindrical robot using a horizontal sweep direction.

of furniture (Figure 6(b)). Each figure also shows the roadmap generated by a cylindrical robot for a particular choice of start and target. As previously noted, the robot constructs only a portion of the roadmap, not the entire roadmap as required in map-building tasks.

Before presenting the experimental results, we introduce a parameter that characterizes the geometrical complexity of the shortest path from S to T. Recall that l_{opt} is the length of the shortest path from S to T in G. Then the *geometrical complexity* of the shortest path is defined as the ratio $\kappa = l_{opt}/\|S-T\|$. Note that $\kappa \geq 1$, and a higher value of κ indicates a more complicated path. The parameter κ is a characteristic of the environment, and we now discuss how to best choose ϵ in terms of κ. To do that, let us consider in detail the initial search performed by the algorithm.

The search begins at the node S, whose global cost is given by $f_{glob}(S) = \|S - T\|$. The robot initially moves along one of the arcs emanating from S, selecting the arc whose direction leads towards the target. Using a local-search behavior endowed by the local cost function f_{loc}, the robot next probes a series of nodes which seems to lead to the target. The nodes participating in this probe are chosen from focal. Since focal is initially given by focal $= \{n : f_{glob}(n) \leq (1+\epsilon)\|S-T\|\}$, the depth of the search probe, measured by the length of the paths leading away from S, is bounded by $(1+\epsilon)\|S-T\|$. The search probe terminates when the only nodes remaining in focal are the open subnodes of S. At this stage the local-search behavior have already brought the robot back to the start node, in order to expand another subnode of S. Continuing in this manner, the robot performs a series of local-search probes until all the subnodes of S are expanded. Only then the minimum value of f_{glob} in open increases, allowing an enlargement of focal and consequently deeper search probes.

Thus the robot initially explores paths whose length from the start node is bounded by $(1+\epsilon)\|S-T\|$. If there exists a path to the target whose length is less than $(1+\epsilon)\|S-T\|$, there is a good chance that the robot will find this path before it expands all the subnodes



(a) (b)

Figure 7: Average travel effort as a function of ϵ. (a) For a a room environment. (b) For an office floor environment. The travel effort is measured in units of length, in environments of size 750×750 units.

of S. The necessary condition for this desirable behavior is:

$$l_{opt} \leq (1+\epsilon)\|S-T\|.$$

In order to satisfy this condition, we have to choose ϵ as follows:

$$\epsilon \geq \frac{l_{opt}}{\|S-T\|} - 1 = \kappa - 1. \tag{2}$$

Thus, *it is desirable to select the parameter* ϵ *larger than* $\kappa - 1$, so that the robot need not return to the start node in order to increase its allowed search depth. Although the robot has no apriori knowledge of κ, many environments possess an evenly distributed geometry, and in such environments κ can be estimated during the search process.

The average travel effort is plotted as a function of ϵ in Figure 7. The graph of Figure 7(a) is for the room environment and the graph of Figure 7(b) is for the office floor environment. Each graph was generated by stepping with ϵ through increments of $\Delta\epsilon = 0.1$ between $\epsilon = 0$ and $\epsilon = 5$. (Only lower values of ϵ are shown in the graphs.) We randomly selected 20 instances of each environment type, and ran in each instance 10 experiments using randomly selected start and target points. Each graph therefore represents 10,000 runs of the algorithm. At the last stage, we smoothed the data points using cubic spline. The resulting graphs have a similar qualitative structure, except for large ϵ values. The graphs can be interpreted as follows. When $\epsilon = 0$, *Roadmap*-A^* becomes a classical A^* algorithm, and the travel effort is very large due to wasteful moves between successively expanded nodes. As ϵ increases, more nodes move into focal and the local-search behavior of the algorithm becomes more pronounced. As a result the travel effort decreases in the lower range of ϵ values. After the initial decrease, there is an intermediate interval of ϵ-values where the two graphs look like a series of hills. This periodic-like behavior reflects the series of search-probes described above, where the robot returns to the vicinity of the start node several times before the search-depth becomes sufficiently large to include the target in its horizon.

As ϵ further increases, there is a sharp fall in the travel effort, when ϵ is large enough to allow the robot to reach the target without returning to the start node. The sharp fall occurs at the *optimal value* $\epsilon^* = 1 - \kappa$ dictated by formula (2). Clearly, we must use a value of ϵ which is larger than this optimal value. Beyond the optimal ϵ-value the two graphs become qualitatively different. In the room environment the travel effort remains flat, while in the office floor environment it grows slowly with ϵ. The reason for this phenomenon is that in the room environment the robot reaches the target using a single local-search probe, which is not affected by increasing ϵ beyond the optimal value. The travel effort in the room environment remains flat even beyond the value of $\epsilon = 2.5$ shown in the graph. In contrast, in the more complicated office floor environment, the local-search probes sometimes drift away from the target and the algorithm imposes an ϵ-dependent dynamic bound on the depth of these search probes. Since a higher value of ϵ means that deeper search probes are allowed, the travel effort increases as ϵ increases in this range of ϵ values. The above discussion indicates that while *we should choose ϵ larger than the optimal ϵ-value for the environment, we must not choose ϵ too much larger, in order to limit the depth of the local-search probes.* Note that while the optimal ϵ-value is not known to the robot, it can be selected with a safety margin and still give a significantly small travel effort.

Algorithm	Scenario		
	room	office floor	maze
A^*	12.067	18.427	14.522
Local-A^* (improved)	1.432	3.652	7.091
Roadmap-A^*	1.432	2.171	3.458
Roadmap-A^* without f_{dir}	4.713	6.852	5.913

Table 1: Average value of l/l_{opt} for the algorithms: A^*, *Local-*A^* (with node splitting and f_{dir}), *Roadmap-*A^*, and *Roadmap-*A^* without f_{dir}. (Note, *Roadmap-*A^* performs better than A^* only in terms of *travel effort*.)

The advantage of *Roadmap-*A^* with suitably chosen ϵ over the *Local-*A^* and A^* algorithms is illustrated in Table 1. To prepare the table, we selected three particular environments: a room environment (Figure 6(a)), an office floor environment (Figure 6(b)), and a third type of environment called maze environment (Figure 9). In each environment we measured the average ratio l/l_{opt} over 100 randomly selected start and target points. For each selection of start and target we ran the following algorithms. The classical A^* algorithm, the local-search algorithm *Local-*A^*, and two versions of the *Roadmap-*A^* algorithm. In the running of *Local-*A^* we added the node splitting technique and the use of f_{dir}. These additions not only reduce the travel effort of *Local-*A^*, but allow us to clearly demonstrate in what ways *Roadmap-*A^* is better than *Local-*A^*. We also measured the performance of *Roadmap-*A^* without the use of f_{dir}, to demonstrate the utility of using the directional selection function. In the running of *Roadmap-*A^*, we computed the optimal ϵ-value for each environment off-line, and used this value as input

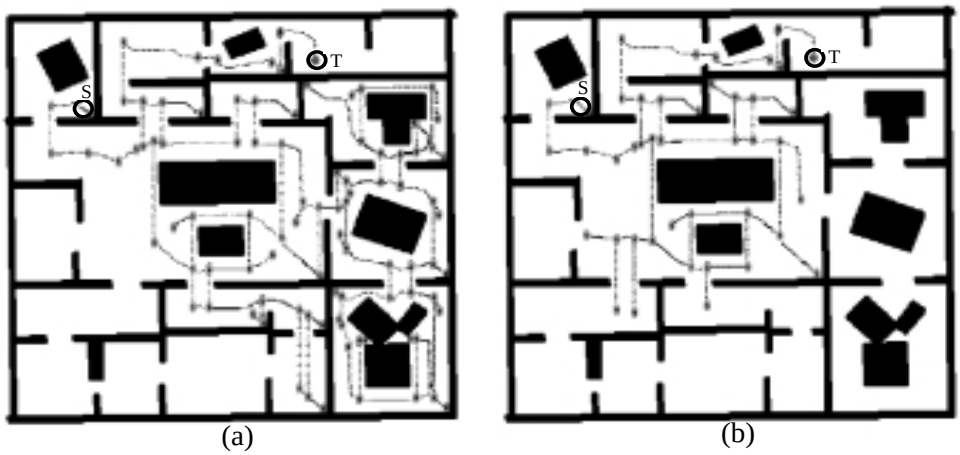


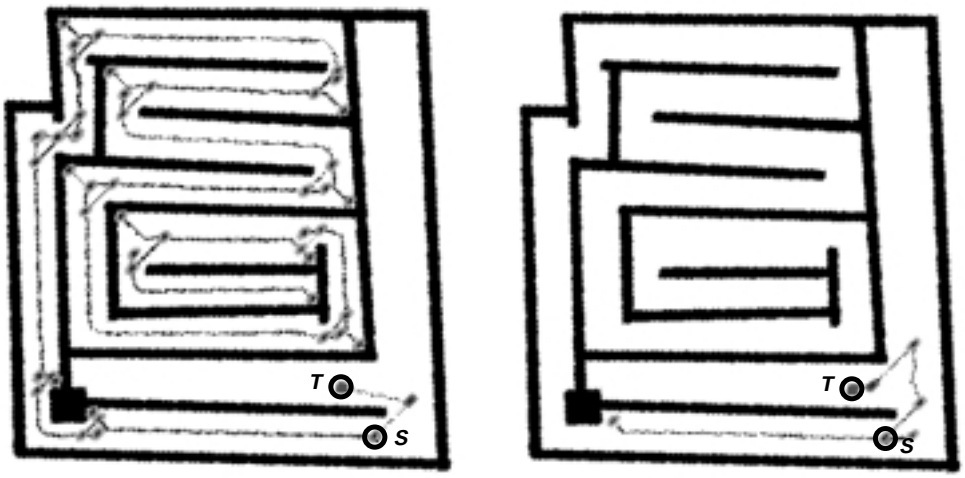
Figure 8: The roadmap generated by the searching robot in an office floor environment, using (a) *Local-A**, and (b) *Roadmap-A**. (Using a horizontal sweep direction.)

to the algorithm. (Such knowledge is not available to the searching robot, but any ϵ not much larger than the optimal ϵ-value would give similar results.)

The results in Table 1 indicate that *Roadmap-A** is much more efficient in terms of travel effort than A^*. The results also indicate that *Roadmap-A** performs similarly to *Local-A** in simple environments. In such environments locally optimal decisions tend to be globally correct, and since we have implemented the *Local-A** algorithm with the travel-effort enhancing features of *Roadmap-A**, the two algorithms exhibit similar performance. However, *Roadmap-A** has an advantage over the *Local-A** algorithm in complex environments, where local-search algorithms often lead the robot away from the target. In particular, *Local-A** performs poorly in the pathological maze environment, while *Roadmap-A** finds the goal rather easily in this environment. Examples of the search path taken by the robot under *Local-A** and *Roadmap-A** in the office floor and maze environments appear in Figures 8 and 9.

6 Concluding Discussion

We presented *Roadmap-A**, an algorithm for searching roadmap graphs constructed online by a robot. The algorithm strives to minimize the travel effort, measured by the length of the path traveled by the robot during the search. At the higher-level *Roadmap-A** runs an A^*_ϵ algorithm, and at the lower-level it runs a *Local-A** algorithm. The A^*_ϵ algorithm maintains a set of nodes called focal, and at every iteration the search agent moves only to a node from this set. The restriction to nodes of focal has the effect of imposing an output-sensitive bound on the depth of the search-probes performed by the lower-level algorithm. At the lower-level, the *Local-A** algorithm moves the search agent through a series of nodes which locally lead to the target and tends to minimize the travel effort associated with moving between successive nodes. An additional mechanism employed by *Roadmap-A** is node splitting, which allows the search agent to expand nodes by tracing



(a) (b)

Figure 9: The roadmap generated the searching robot in a maze environment, using (a) *Local*-A^*, and (b) *Roadmap*-A^*. (Using a sweep direction inclined 45° below the horizon.)

one arc at a time. The node splitting technique allows the search agent to move in an efficient depth-first fashion, while still keeping the entire operation of *Roadmap*-A^* within the framework of the A^*_ϵ algorithm.

We also presented several formal properties of *Roadmap*-A^*, inherited by its membership in the A^*_ϵ family of algorithms. In particular, the search space of *Roadmap*-A^* is bounded by an ellipse of size $(1+\epsilon)l_{opt}$, where l_{opt} is the length of the shortest path from S to T. In the case where the nodes are evenly distributed, the ellipse also bounds the travel effort in a way which depends on l_{opt}, ϵ, and $\|S-T\|$. We also measured the average travel-effort as a function of ϵ in simulated environments. In the lower-range of ϵ values there is a steep drop in the travel effort, due to the increasing influence of the local-search component in the behavior of the algorithm. Then, after an intermediate range of ϵ-values, there is another sharp drop in the travel effort at the optimal ϵ-value. The optimal ϵ-value is determined by the ratio $\kappa = l_{opt}/\|S-T\|$, and the results indicate that the travel effort in the interval just beyond the optimal ϵ-value is still very low. Hence in practice we only need to select ϵ with a safety margin beyond the optimal value in order to get significantly low travel effort.

Following are several issues for further research. First, it is possible for a mobile robot to sense portions of the roadmap arcs emanating from its current location. Rather then use such additional information, we made the conservative assumption that the agent can only sense the direction of the arcs emanating from its current position. Our approach is formally justified, as it is known that even simple environments can render powerful sensors myopic. Furthermore, *Roadmap*-A^* can also be used for searching roadmaps which represent the free configuration space of mechanisms other than mobile robots, such as robot arms. For such mechanisms it is not immediately clear how sensors can

provide information about the roadmap arcs. Second, while discussing the ellipse bound, we mentioned a technique for introducing intermediate nodes at regular intervals of length l_{arc}. This technique ensures that *every* node visited by the search agent would reside in an ellipse of size $(1+\epsilon)l_{opt}+2l_{arc}$. However, how to best select the parameter l_{arc}, preferably in an output-sensitive way, is an open issue.

One last issue for further research is the derivation of worst-case[6] lower bounds on the travel effort associated with on-line search by a robot equipped with sensors. Knowledge of such bounds would enable us to assess in an absolute way the quality of a given on-line navigation algorithm [20][p. 806-808]. It is worth mentioning that some results on such universal bounds are already available in the literature. Lumelsky [12], Sankaranarayanan [21], and their coworkers derive such bounds for a point robot moving in the plane. However, their bounds are expressed only in terms of the obstacles' perimeter and the distance $\|S-T\|$, not in terms of the crucial parameter l_{opt}. Berman [1], Blum [2], Papadimitriou [14], and their coworkers derive lower bounds on the search effort which explicitly depend on l_{opt}. However, they discuss only special types of environments. The derivation of worst-case lower bounds for general cases, such as the case of searching general roadmaps discussed in this paper, is an open problem.

Acknowledgments: The authors would like to acknowledge the insightful comments made by Richard Korf and Shaul Markovitch on early versions of this paper.

A Review of the A^*_ϵ Algorithm

The A^*_ϵ algorithm is a variant of the classical A^* algorithm [16], which uses in addition to the open and close sets of nodes, a subset of open called focal. The focal set is defined as follows,

$$\mathsf{focal} = \left\{ n \in \mathsf{open} : f(n) \leq (1+\epsilon) \min_{m \in \mathsf{open}} \{f(m)\} \right\},$$

where $\epsilon > 0$ is a parameter. The A^*_ϵ algorithm operates identically to A^*, except that it chooses for expansion the node from focal with a minimal F-value, where F is a second evaluation function. The function f has the additive structure of A^*, given by $f(n) = g(S,n) + h(n,T)$, where $g(S,n)$ is the length of the path from S to n, and $h(n,T)$ an estimate of the length of the path from n to T. In contrast, the function F can have any desired form and may use any information available to the search agent. The parameter ϵ can be interpreted as controlling the relative emphasize on the functions f and F. In particular $\epsilon = 0$ gives the classical A^* algorithm, while $\epsilon = \infty$ gives a best-first-search algorithm which is wholly controlled by the function F.

The A^*_ϵ algorithm has properties similar to those of A^*. Like A^*, it terminates on finite graphs and is complete on possibly infinite graphs. The solution found by A^*_ϵ is ϵ-optimal, as made precise in the following theorem.

[6] That is, for any on-line search algorithm, there exists an obstacle course where the algorithm generates a path at least that long.

Theorem 1 ([15],Theorem 3.13) *Let the function $h(n,T)$ be admissible. When A^*_ϵ selects a node z for expansion, the f-value of z satisfies $f(z) \leq (1+\epsilon)l_{opt}$, where l_{opt} is the length of the shortest path from S to T.*
In particular, A^*_ϵ always finds a path from S to T whose length l satisfies $l \leq (1+\epsilon)l_{opt}$.

Last we state the ellipse property of A^*_ϵ, which is a straightforward extension of a similar property of A^* [15, p. 146-150]. We assume that the function $h(n,T)$ is given by the Euclidean distance, $h(n,T) = \|n-T\|$, although the result can be generalized to any admissible function $h(n,T)$.

Theorem 2 *Let a graph G be embedded in Euclidean space with the cost of edges given by their length, and let $h(n,T) = \|n-T\|$. Then when A^*_ϵ selects a node z for expansion, the node satisfies*

$$\|z-S\| + \|z-T\| \leq (1+\epsilon)l_{opt},$$

where l_{opt} is the length of the shortest path from S to T. Thus A^*_ϵ expands only those nodes which lie in an ellipse with focal points S and T and size $(1+\epsilon)l_{opt}$.

Proof: The function f is given by $f(n) = g(S,n) + \|n-T\|$. Since $h(n,T) = \|n-T\|$ is admissible, we may invoke Theorem 1. According to the theorem, when a node z is selected for expansion from focal it satisfies $f(z) = g(S,z) + \|z-T\| \leq (1+\epsilon)l_{opt}$. On the other hand, the length of the path from S to z along the graph is equal or larger than the Euclidean distance between S and z. Hence $g(S,z) \geq \|z-S\|$, and consequently $\|z-S\| + \|z-T\| \leq (1+\epsilon)l_{opt}$. □

B Implementation of the *IRoadmap* Method

To implement the *IRoadmap* method on a cylindrical mobile robot, we need to specify how to trace a ridge-curve, how to detect a ridge-curve endpoint, and how to implement the critical point detector. First we consider the tracing of a ridge-curve. The robot's range sensor induces an imaginary ring surrounding the robot, whose radius is equal to the distance of the robot to the closest obstacle. Using the range data as a repulsive field, the robot moves away from the closest obstacle while limiting its position to a slice perpendicular to the sweep direction. When the distance to the obstacles achieves a maximum, the robot has reached a ridge-curve. The robot then traces the ridge-curve by moving along the sweep direction by a small increment, then repeating the distance maximization process in the new slice. More sophisticated tracing techniques are also available, see e.g. [5].

Next we describe how the robot detects the endpoints of a ridge-curve. An endpoint which lies on the boundary of the freespace is easily detectable, since the robot touches some physical obstacle at this point. A bifurcation endpoint is detected with the technique depicted in Figure 5(a). Let the *central line* of the robot be the line passing through the center of the robot parallel to the sweep direction. When the robot is located in the

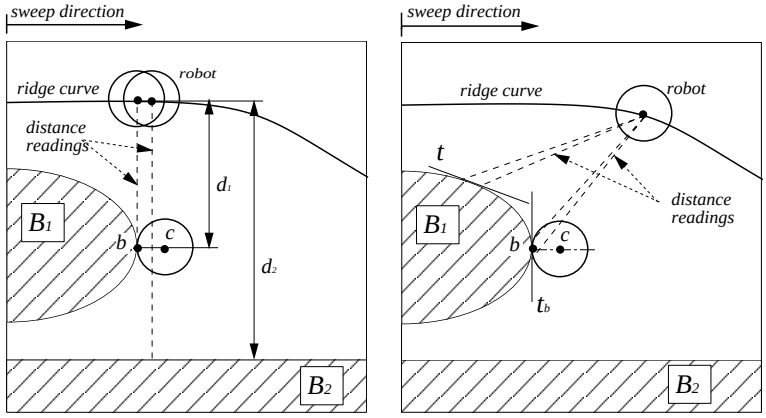


Figure 10: Detection of critical points by (a) distance discontinuity, (b) obstacle tangent perpendicular to the sweep direction.

interior of a ridge-curve, its range sensor detects the two closest obstacles on both sides of the robot's central line. At a bifurcation point, the range-sensor's ring touches one of the two closest obstacles exactly on the central line. At this location the two closest obstacles are on the same side of the robot, and they repulse the robot towards another ridge-curve. This is precisely the condition for a bifurcation endpoint.

Last we describe the implementation of the critical point detector. It can be verified that the critical points in c-space lie on the boundary of the c-obstacles, at points where the tangent to the c-obstacle is perpendicular to the sweep direction. It can further be verified that the normal to the boundary of a physical obstacle $\mathcal{B}_i$ at b is also normal to the boundary of the corresponding c-obstacle $\mathcal{CB}_i$ at a point c where the normal crosses the boundary of $\mathcal{CB}_i$ (Figure 5(b)). Thus the tangent to $\mathcal{B}_i$ at b is parallel to the tangent to $\mathcal{CB}_i$ at c. The robot detects the critical points by measuring the distance to the obstacles along two rays which are perpendicular to the sweep direction (Figure 10(a)). If one of the rays detects a distance-discontinuity between successive readings, the point of discontinuity, denoted b, lies on the boundary of an obstacle $\mathcal{B}_i$. Assuming that the robot coordinate r_x is aligned with the sweep direction, the coordinates of b are: $b = (r_x, r_y + min\{d_i, d_{i+1}\})$, where d_i and d_{i+1} are the successive distance readings and (r_x, r_y) are the robot coordinates. Since the measurement ray is tangent to $\mathcal{B}_i$ at b, the tangent to $\mathcal{CB}_i$ at the corresponding point c is perpendicular to the sweep direction, and c is a critical point. If $d_i < d_{i+1}$ the critical point is a join point, and if $d_i > d_{i+1}$ it is a split point. The c-space coordinates of c are: $c = (b_x \pm R, b_y)$, where $b = (b_x, b_y)$ is the point on the boundary of $\mathcal{B}_i$, R is the radius of the robot, and the $\pm$ sign depends on the direction of motion of the robot and its position with respect to the obstacle.

In our implementation, we have found that practical range sensors such as sonars obtain exact readings only when the measurement ray is nearly perpendicular to an obstacle boundary. But by construction at a critical point the measurement ray is tangent to an

obstacle boundary. Hence we augment the distance-discontinuity measurement with the following predictive technique. Each time the robot reaches a new node on the roadmap, it scans the environment for obstacle boundary points whose tangent is perpendicular to the sweep direction (Figure 10(b)). These points satisfy a necessary condition for a critical point. When the robot passes through the slice of such a point it check for distance discontinuity at this point. The combination of the predictive technique with the distance-discontinuity check provides a reasonably accurate detection of the critical points in practice.

References

[1] P. Berman, A. Blum, A. Fiat, H. Karloff, A. Rosen, and M. Saks. Randomized robot navigation algorithms. In *SODA*, pages 75–84, 1996.

[2] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar terrain. In *STOC*, pages 494–504, 1991.

[3] J. F. Canny and M. C. Lin. An opportunistic global path planner. *Algorithmica*, 10:102–120, 1993.

[4] R. Chatila. Deliberation and reactivity in autonomous mobile robots. *Robotics and Autonomous Systems*, 16:197–211, 1995.

[5] H. Choset and J. W. Burdick. Sensor based planning and nonsmooth analysis. In *IEEE Int. Conf. on Robotics and Automation*, pages 3034–3041, 1994.

[6] H. Choset and J. W. Burdick. Sensor based planning, part ii: Incremental construction of the generalized voronoi graph. In *IEEE Int. Conference on Robotics and Automation*, pages 1643–1649, 1995.

[7] G. Foux, M. Heymann, and A. Bruckstein. Two dimensional robot navigation among unknown stationary polygonal obstacles. *IEEE Trans. on Robotics and Automation*, 9(1):96–102, 1993.

[8] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. System Science and Cybernatics*, SSC-4(2):100–107, 1968.

[9] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[10] R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[11] V. J. Lumelsky. Continuous robot motion planning in unknown environment. In K. S. Narendra, editor, *Adaptive and Learning Systems*, pages 339–358. Plenum, NY, 1986.

[12] V. J. Lumelsky and A. Stepanov. Path planning strategies for point automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[13] H. Noborio and T. Yoshioka. An on-line and deadlock-free path-planning algorithm based on world topology. In *Conf. on Intelligent Robots and Systems, IROS*, pages 1425–1430. IEEE/RSJ, 1993.

[14] C. H. Papadimitriou and M. Yanakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[15] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA, 1984.

[16] J. Pearl and J. H. Kim. Studies in semi-admissible heuristics. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(4), July 1982.

[17] N. S. V. Rao and S. S. Iyengar. Autonomous robot navigation in unknown terrains: visibility graph based methods. *IEEE trans. on Systems, Man and Cybernetics*, 20(6):1443–1449, 1990.

[18] E. Rimon. Construction of c-space roadmaps from local sensory data: What should the sensors look for? *Algorithmica*, 17(4):357–379, 1997.

[19] A. Rosenfeld, E. Rivlin, and S. Khuller. Learning to navigate on a graph. In *Image Understanding Workshop*, 1995.

[20] S. J. Russel and P. Norvig. *Artificial Intelligence, A Modern Approach.* Prentice-Hall, 1995.

[21] A. Sankaranarayanan and M. Vidyasagar. Path planning for moving a point object amidst unknown obstacles in a plane: the universal lower bound on worst case path lengths and a classification of algorithms. In *IEEE Int. Conf. on Robotics and Automation*, pages 1734–1941, 1991.

[22] Z. Shiller. On-line sub-optimal obstacle avoidance. In *IEEE Int. Conf. on Robotics and Automation*, pages 335–340, 1999.

[23] A. Stentz. Optimal and efficient path planning for partially known environments. In *IEEE Int. Conf. on Robotics and Automation*, pages 3310–3317, 1994.

[24] C. J. Taylor and D. J. Kriegman. Vision based motion planning and exploration algorithms for mobile robots. In *Workshop on Algorithmic Foundations of Robotics*, pages 69–83. A K Peters, 1995.