IN *Proc. Int. Workshop on the Algorithmic Foundations of Robotics, 2008*

Bounded Uncertainty Roadmaps for Path Planning

Leonidas J. Guibas[1], David Hsu[2], Hanna Kurniawati[2], and Ehsan Rehman[2]

[1] Department of Computer Science, Stanford University, USA
guibas@cs.stanford.edu
[2] Department of Computer Science, National University of Singapore, Singapore
{dyhsu,hannakur,ehsanreh}@comp.nus.edu.sg

Abstract. Motion planning under uncertainty is an important problem in robotics. Although probabilistic sampling is highly successful for motion planning of robots with many degrees of freedom, sampling-based algorithms typically ignore uncertainty during planning. We introduce the notion of a *bounded uncertainty roadmap* (BURM) and use it to extend sampling-based algorithms for planning under uncertainty in environment maps. The key idea of our approach is to evaluate uncertainty, represented by collision probability bounds, at multiple resolutions in different regions of the configuration space, depending on their relevance for finding a best path. Preliminary experimental results show that our approach is highly effective: our BURM algorithm is at least 40 times faster than an algorithm that tries to evaluate collision probabilities exactly, and it is not much slower than classic probabilistic roadmap planning algorithms, which ignore uncertainty in environment maps.

1 Introduction

Probabilistic sampling is a highly successful approach for motion planning of robots with many degrees of freedom. Sampling-based motion planning algorithms typically assume that input environments are perfectly known in advance. This assumption is reasonable in carefully engineered settings, such as robot manipulators on manufacturing assembly lines. As robots venture into new application domains at homes or in offices, environment maps are often acquired through sensors subject to substantial noise. It is essential for sampling-based motion planning algorithms to take into account uncertainty in environment maps during planning so that the resulting motion plans are relevant and reliable.

In sampling-based motion planning, a robot's configuration space $\mathcal{C}$ is assumed to be known and represented implicitly by a geometric primitive $\texttt{Free}(q)$, which returns true if and only if the robot placed at q does not collide with obstacles in the environment. The main idea is to capture the connectivity of $\mathcal{C}$ in a graph, usually called a *roadmap*. The nodes of the roadmap correspond to collision-free configurations sampled randomly from $\mathcal{C}$ according to a suitable probability distribution. There is an edge between two nodes if the straight-line path between them is collision-free. Now suppose that the shapes or poses of obstacles in the environment are not known exactly, but are modeled as a probability distribution π_c of possible shapes and poses.

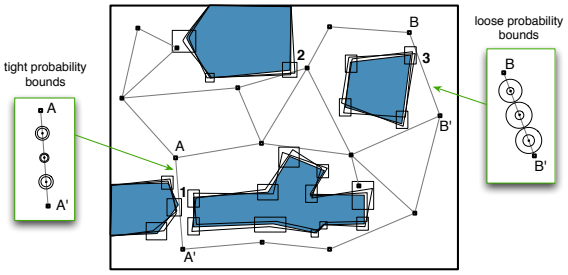


Fig. 1. An uncertainty roadmap. The positions of polygon vertices are uncertain and modeled as probability distributions. The rectangular boxes around the vertices indicate the regions of uncertainty. In the insets, for each configuration marked along a roadmap edge, there are two circles indicating the upper and lower bounds on the probability that the configuration is collision-free. The size of the circles indicates the probability value. In region 1, the two circles have similar size, indicating that the probability bounds are tight.

Then $\texttt{Free}(q)$ cannot always determine whether q is collision-free or not: it depends on the distribution of obstacle poses and shapes. Instead of relying on $\texttt{Free}(q)$, we need to compute the probability that q is collision-free with respect to π_c, and instead of a usual roadmap, we construct an *uncertainty roadmap* U by annotating roadmap edges with probabilities that they are collision-free (Fig. 1). We then process path planning queries by finding a path in U that is best in the sense of low collision probability or other suitable criteria that take into account, *e.g.*, path length as well.

Unfortunately, constructing a complete uncertainty roadmap U incurs high computational cost, as computing collision probabilities exactly is very expensive computationally. It effectively requires integrating over a high-dimensional distribution π_c of obstacle shapes and poses. The *dimensionality* of π_c depends on the geometric complexity of the environment obstacles. Consider, for example, a simple two-dimensional environment consisting of 10 line segments. Each line segment is specified by its endpoints, whose positions are uncertain. We then need to integrate over a distribution of $10 \times 2 \times 2 = 40$ dimensions!

To overcome this difficulty, we turn the high-dimensional integral into a series of lower-dimensional ones. More interestingly, observe that a path planning query may be answered without knowing the complete uncertainty roadmap with exact collision probabilities. We maintain upper and lower bounds on the probabilities and refine the bounds incrementally as needed. We call such a roadmap a *bounded uncertainty roadmap* (BURM). See the insets in Fig. 1 for an illustration. The key idea of our approach is to evaluate uncertainty, represented by the collision probability bounds, at multiple resolutions in different regions of the configuration space, depending on their relevance for finding a best path in U. Often, the critical decision of favoring one path over another depends on the uncertainty in localized regions only, for example, in narrows passages where the robot must operate in close proximity of the obstacles. It is thus sufficient to evaluate uncertainty accurately only in those regions and only to the extent necessary to choose a best path. Consider the example in Fig. 1. If we want to go form A to A', it is important to evaluate the precise collision probabilities

in region 1 in order to decide whether to take the risk of going through the narrow passage or to make a detour. Knowing the precise collision probabilities in regions 2 and 3 is much less relevant for this decision. Thus, evaluating collision probabilities at different resolutions hierarchically leads to drastic reduction in computation time by avoiding unnecessarily computing the exact collision probabilities.

In the following, we start by briefly reviewing related work (Section 2). We then introduce the notion of a BURM (Section 3) and describe our approach for path planning with BURMs (Section 4). Preliminary experimental results show that our approach for path planing under uncertainty is highly effective: in our tests, it is at least 40 times faster than an algorithm that tries to evaluate collision probabilities exactly, and it is not much slower than classic probabilistic roadmap planning algorithms, which ignore environment uncertainty (Section 5). Finally, we conclude with directions for future work (Section 6).

2 Related Work

Motion planning under uncertainty is an important problem in robotics and has been studied widely [13, 14, 22]. In robot motion planning, uncertainty arises from two main sources: (i) noise in robot control and sensing and (ii) imperfect knowledge of the environment. In this work, we address the second only. Partially observable Markov decision processes (POMDPs) is a general and principled approach for planning under uncertainty [20, 10]. Unfortunately, under standard assumptions, the computational cost of solving a POMDP exactly is exponential the number of states of the POMDP [18]. An uncertain environment usually generates a large number of states. Despite the recent advances in approximate POMDP solvers [19, 21, 12], uncertain environments still pose a significant challenge for POMDP planning. In mobile robot motion planning, a common representation of an uncertain environment is an occupancy grid. Each cell of an occupancy grid contains the probability that the cell is occupied by obstacles. Assuming that uncertainty in robot control and sensing is negligible, one can find a path with minimum expected collision cost by graph search algorithms, such as Dijkstra's algorithm or the A* algorithm. In the motion planning literature, occupancy-grid planning belongs to the class of approximate cell decomposition algorithms [13], whose main disadvantage is that they do not scale up well as a robot's number of degrees of freedom increases.

Probabilistic sampling of the robot's configuration space is the most successful approach for overcoming this scalability issue [5, 8, 14]. Our work is based on this approach, but extends it to deal with uncertainty in environment maps. Usually, sampling-based motion planning algorithm first build a roadmap that approximates the connectivity of the robot's configuration space and then search for a collision-free path in the roadmap. The idea that a path planning query can be answered without constructing the complete roadmap in advance is a form of lazy evaluation and has appeared before in sampling-based algorithms for single-query path planning, *e.g.*, Lazy-PRM [2], EST [9], and RRT [15]. However, since classic motion planning assumes perfect knowledge of the geometry of the robot and the obstacles, lazy construction of the roadmap is simpler.

Sampling-based motion planning has been extended to deal various types of uncertainty, including robot control errors [1], sensing errors [4, 23], and imperfect environment maps [17]. Our problem is related to that in [17]. To overcome the difficulty of computing collision probability, the earlier work proposes a nearest-point approximation technique. Although the approximation is supported by experimental evidence, its error is difficult to quantify. Also, the technique is restricted to two-dimensional environments only [17].

3 Bounded Uncertainty Roadmaps

Let us start with two-dimensional environments. The obstacles are modeled as polygonal objects, each consisting of a set of primitive geometric features—line segments for two-dimensional environments. The endpoints of the line segments are not known precisely and modeled as probability distributions with finite support, such as truncated Gaussians. See Fig. 1 for an illustration. Environment maps of this kind can be obtained by, for example, feature-based extended Kalman filtering (EKF) mapping algorithms [22]. For generality, we model the robot in exactly the same way. In three-dimensional environments, the representation is similar, but the primitive geometric feature are triangles rather than line segments.

Given such a representation of the obstacles and the robot, we can construct an uncertainty roadmap U in the robot's configuration space $\mathcal{C}$. The nodes of U are configurations sampled at random from $\mathcal{C}$. For every pair of nodes u and u' that are close enough according to some metric, there is an edge in U, representing the straight-line path between u and u'. Recall that in classic motion planning, sampling-based algorithms construct a roadmap whose nodes and edges are guaranteed to be collision-free, and the goal is to find a collision-free path in the roadmap. In our setting, due to the uncertainty, we cannot guarantee that the nodes and edges of U are collision-free, and there may exist no path that is collision-free with probability 1. So instead, we want to find a path with minimum cost according to a suitable cost function. A cost function may incorporate various properties of the desired path. To be specific, our cost function incorporates two considerations: the collision probability and the path length. This allows us to trade off the distance that the robot must travel against the risk of collision.

To define such a path cost function, we assign a weight to each edge e of U:

$$W(e) = \ell(e) + \mathbf{E}[C(e)], \tag{1}$$

where $\ell(e)$ is the length of e and $C(e)$ is the cost of collision for e. $\mathbf{E}[C(e)]$ denotes the expected collision cost, and the expectation is taken over π_c, the probability distribution of the obstacle and robot geometry. This cost function assumes that collision is tolerable and we want to trade off the risk of collision against the robot's travel distance. Paths that do not conform to this assumption, *e.g.*, those that penetrate through the interior of obstacles, must be excluded. To obtain $W(e)$, we need to calculate $\mathbf{E}[C(e)]$. Doing so directly is extremely difficult, because of the need to integrate over π_c, a high-dimensional distribution whose dimensionality is proportional to the number of geometric features describing the obstacles and the robot, as

illustrated by the example in Section 1. Furthermore, we must perform this integration for every edge of U. Instead of this, we break down the integration process into several steps. First, we integrate over the configurations contained in e:

$$C(e) = \int_{q \in e} C(q)\, dq,$$

where we slightly abuse the notation and use $C(q)$ to denote the collision cost at q. Following the usual practice in sampling-based motion planning, we discretize the edge e into a sequence of of configurations $(q_1, q_2, \ldots, q_n)$ at a fixed resolution and approximate the integral by

$$C(e) = \sum_{i=1}^{n} C(q_i). \tag{2}$$

Next, recall that the obstacles and the robot are each represented as a polygonal object consisting of a set of primitive geometric features. Denote the two feature sets by S for the obstacles and S' for the robot. We define the collision cost of the robot with the obstacles as the sum of collision costs of all pairs of geometric features $s \in S$ and $s' \in S'$. This can model, for example, the preference that configurations with fewer feature pairs in collision are more desirable. In formula, we have

$$C(q) = \sum_{s \in S, s' \in S'} C_{s,s'}(q), \tag{3}$$

where $C_{s,s'}(q)$ is the collision cost for the feature pair s and s' when the robot is placed at configuration q. Combining Eqs. (1–3) and using the linearity of expectation, we get $W(e) = \ell(e) + \sum_{i=1}^{n} \sum_{s \in S, s' \in S'} \mathbf{E}[C_{s,s'}(q_i)]$. Let $I_{s,s'(q)}$ denote the event that s and s' intersect when the robot is placed at q. Then $\mathbf{E}[C_{s,s'}(q_i)] = \alpha\, \mathrm{P}(I_{s,s'(q)})$, where α is the cost of collision when a pair of features intersect. The value of α is usually constant for two-dimensional environments, but may vary according to s and s' for three-dimensional environments. In practice, α is adjusted to reflect our willingness to take the risk of collision in order to shorten the robot's travel distance. To summarize, the weight of an edge e is given by

$$W(e) = \ell(e) + \sum_{i=1}^{n} \sum_{s \in S, s' \in S'} \alpha\, \mathrm{P}(I_{s,s'(q_i)}), \tag{4}$$

and the cost of a path γ in U is the sum of the weights of all edges contained in γ.

To compute the cost of a path, each edge of an uncertainty roadmap U must carry a set of probabilities $\mathrm{P}(I_{s,s'(q_i)})$. Although s and s' are primitive geometric features of constant size, computing $\mathrm{P}(I_{s,s'(q_i)})$ exactly is still expensive. If s and s' are both uncertain, then the computation requires integration over a distribution of 8 dimensions for two-dimensional features and 18 dimensions for three-dimensional features. To reduce the computational cost, we maintain upper and lower bounds on $\mathrm{P}(I_{s,s'(q_i)})$ rather than calculate the exact probability. Thus each edge e of U carries a set of probability bounds on $\mathrm{P}(I_{s,s'(q_i)})$ for each configuration $q_i \in e$ resulting from the discretization of e and for each pair of features $s \in S$ and $s' \in S'$. We call such a roadmap a *bounded uncertainty roadmap* or BURM for short. The probability bounds are refined incrementally by subdividing the integration domain hierarchically during the path finding.

4 Path Planning with BURMs

4.1 Overview

Suppose that we are given the (uncertain) geometry of the obstacles and the robot in the representation described in the previous section. Our goal is to find a minimum-cost path between a start configuration q_{s} and a goal configuration q_{g}. Conceptually, there are two steps, First, we construct a BURM U with trivial probability bounds by sampling the robot's configuration space $\mathcal{C}$. Next, we tighten up the probability bounds incrementally while searching for a minimum-cost path in U.

The first step is similar to that in the usual sampling-based motion planning algorithms. We sample a set of configurations from $\mathcal{C}$ according to a suitable probability distribution and insert the sampled configurations along with q_{s} and q_{g} as nodes of U. We then create an edge for every pair of nodes that are sufficiently close according to some metric. We filter out those nodes and edges that are in collision. Here collision is defined with respect to the mean geometry of the obstacles and the robot, which means that the primitive geometric features representing the obstacles and the robot are all at their mean positions. The purpose of filtering is to exclude those paths that cause the robot to pass through the interior of the obstacles. It is well known that the probability distribution for sampling $\mathcal{C}$ is crucial, and there is a lot of work on effective sampling strategies for motion planning. See [5, 8, 14] for comprehensive surveys. There is also recent work on how to adapt the sampling distribution when the environment map is uncertain. In this paper, we do not address the issue of sampling strategies. BURMs can be used in combination with any of the existing sampling strategies.

In the second step, we search for a minimum-cost path in U using a variant of Dijkstra's algorithm. While Dijkstra's algorithm deals with path cost, a BURM contains only bounds on path cost. When there are two alternative paths, we may not be able to decide which one is better, as their bounds may "overlap". To resolve this, we need to refine the probability bounds in a suitable way. The details are described in the next three subsections.

4.2 Searching for a Minimum-Cost Path

Given a BURM U, we search for a minimum-cost path in U using a variant of Dijkstra's algorithm. A sketch of the algorithm is shown in Algorithm 1. For each node u in U, we maintain the lower bound $\underline{K}(u)$ and upper bound $\overline{K}(u)$ on the minimum-cost path from q_{s} to u. Recall that every edge e of U carries a set of probability bounds on $I_{s,s'(q_i)}$, for every $q_i \in e$ resulting from the discretization of e and every feature pair $s \in S$ and $s' \in S'$. Let $\underline{\mathrm{P}}(I_{s,s'(q_i)})$ and $\overline{\mathrm{P}}(I_{s,s'(q_i)})$ denote the lower and upper bounds on the probability of $I_{s,s'(q_i)}$, respectively. Using these bounds, we can calculate the lower bound $\underline{W}(e)$ and upper bound $\overline{W}(e)$ on the edge weight for each edge $e \in U$. By the definition, $\underline{K}(u)$ and $\overline{K}(u)$ can then be obtained by summing up the bounds on the edge weights. Like Dijkstra's algorithm, we insert each node u of U into a priority queue Q_u, which is implemented as a heap, and then dequeue them one by one until a minimum-cost path to q_{g} is found. However, instead of using the path cost from q_{s} to u as the priority value, we use the path cost bounds

Algorithm 1 Searching for a minimum-cost path in a BURM.

1: For every node u of a BURM U, initialize the lower and upper bounds on the cost of the minimum-cost path from q_{s} to u: $\underline{K}(u) = 0, \overline{K}(u) = 0$ if $u = q_{\mathrm{s}}$, and $\underline{K}(u) = +\infty, \overline{K}(u) = +\infty$ otherwise.
2: Insert all nodes of U into a priority queue Q_u.
3: **while** Q_u is not empty **do**
4: Find in Q_u a node u such that $\overline{K}(u) \leq \underline{K}(v)$ for all $v \in Q_u$. Remove u from Q_u.
5: **if** u = q_{g} **then return**.
6: **for** every node v incident to u **do**
7: Discretize the edge between u and v at a given resolution into a sequence of configurations $q_i, i = 1, 2, \ldots$.
8: For every q_i, invoke `FindColEvents`(q_i, S, S') to find feature pairs $s \in S$ and $s' \in S'$ that are likely to have $\mathrm{P}(I_{s,s'(q_i)}) > 0$. For each such feature pair, set $\underline{\mathrm{P}}(I_{s,s'(q_i)}) = 0$ and $\overline{\mathrm{P}}(I_{s,s'(q_i)}) = 1$, and insert $I_{s,s'(q_i)}$ into Q_{e}.
9: Set $\underline{K}_u(v) = \underline{K}(u) + \underline{W}(u, v)$ and $\overline{K}_u(v) = \overline{K}(u) + \overline{W}(u, v)$.
10: **while** the two intervals $(\underline{K}_u(v), \overline{K}_u(v))$ and $(\underline{K}(v), \overline{K}(v))$ overlap **do**
11: `RefineProbBounds`$(Q_{\mathrm{e}}, U, q_{\mathrm{s}}, u, v)$.
12: **if** $\overline{K}_u(v) < \underline{K}(v)$ **then**
13: Set $\underline{K}(v) = \underline{K}_u(v)$ and $\overline{K}(v) = \overline{K}_u(v)$.
14: Update Q_u, using the new bounds on the cost of the minimum-cost path to v. Call `RefineProbBounds` if needed.

$(\underline{K}(u), \overline{K}(u))$. For two nodes u and v in U, We say that u has higher priority than v, if $\overline{K}(u) \leq \underline{K}(v)$. In our implementation of Q_u, if the bound intervals $(\underline{K}(u), \overline{K}(u))$ and $(\underline{K}(v), \overline{K}(v))$ overlap, we refine the probability bounds until we can establish which node has higher priority in Q_u.

As we have mentioned in Section 3, computing collision probabilities requires integration over a high-dimensional distribution and is very expensive computationally. We use two techniques for efficient computation of the probability bounds. In line 8 of Algorithm 1, `FindColEvents` find feature pairs $s \in S$ and $s' \in S'$ such that s and s' are likely to intersect with non-zero probability, when the robot is placed at q_i. For each such feature pair, we attach an initial probability bound of $[0, 1]$ to the event $I_{s,s'(q_i)}$ and insert $I_{s,s'(q_i)}$ into a set Q_{e} as a candidate for probability bound refinement in the future. Observe that usually, at each configuration, only a small number of feature pairs are in close proximity and likely to intersect with non-zero probability. Therefore, this step drastically reduce the number of collision probability bounds that need to be calculated. To search for the intersecting feature pairs efficiently, we exploit a hierarchical representation of the geometry of the obstacles and the robot (see Section 4.3) and quickly eliminate most of the feature pairs that are guaranteed to have zero collision probability.

In lines 11 and 14 of Algorithm 1, `RefineProbBounds` refines probability bounds. While searching for a minimum-cost path in U, we may encounter two paths γ and γ' and must decide which one has lower cost. If the bound intervals on the cost of γ and γ' overlap, refinement of the probability bounds becomes necessary. To do so, we find all events $I_{s,s'(q)}$ in Q_{e} such that q lies in an edge along γ or γ'. We then refine the probability bounds on these events (see Section 4.4), until we can

determine the path with lower cost. For efficiency, we order the events found with a heuristic and process those more likely to help separate the bound intervals first.

To focus on the main issue and keep the presentation simple, Algorithm 1 uses Dijkstra's algorithm for graph search. Informed search, such as the A* algorithm with an admissible heuristic function, is likely to give better results. In our case, one possible heuristic function is the Euclidean distance between two configurations.

4.3 Bounding Volume Hierarchies

In this and next subsections, we describe our computation of probability bounds. We restrict ourselves to the two-dimensional case. The basic idea generalizes to the three-dimensional case in a straightforward way, but the details are more involved.

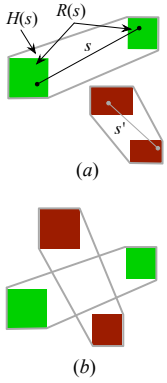


Fig. 2. The line segment pair s and s' intersect with (*a*) probability 0 and (*b*) probability 1.

For the two-dimensional case, `FindColEvents` (Algorithm 1, line 8) tries to find line segment pairs $s \in S$ and $s' \in S'$ that are likely to have intersection probability $\mathrm{P}(I_{s,s'(q)}) > 0$, when s' is placed at configuration q. It does so by quickly eliminating most of the line segment pairs with $\mathrm{P}(I_{s,s'(q)}) = 0$. Recall that we model the endpoints of these line segments as probability distributions with finite support. Without loss of generality, assume that the support regions are rectangular. Let $R(s)$ denote the endpoint regions for a line segment s and $H(s)$ denote the convex hull of $R(s)$. Using the result below, we can check whether a line segment pair s and s' has $\mathrm{P}(I_{s,s'(q)}) = 0$ in constant time, as the convex hulls and the endpoint regions of s and s' are all polygons with a constant number of edges. See Fig. 2 for an illustration.

Theorem 1. *Let s and s' denote two line segments with uncertain endpoint positions in two dimensions.*

(1) *If $H(s) \cap H(s') = \emptyset$, then s and s' intersect with probability* 0.
(2) *If $H(s) \cap H(s') \neq \emptyset$, $R(s) \cap H(s') = \emptyset$, and $R(s') \cap H(s) = \emptyset$, then s and s' intersect with probability* 1.

If S and S' contain m and n line segments, respectively, it takes $O(mn)$ time to check all line segment pairs $s \in S$ and $s' \in S'$. This is very time-consuming, as we need to invoke `FindColEvents` repeatedly at many configurations. To improve efficiency, we apply a well-known technique from the collision detection literature and build bounding volume hierarchies over the geometry of the obstacles and the robot. There are many different types of bounding volume hierarchies. See [16] for a survey. We have chosen the sphere tree hierarchy, though other hierarchies, such as the oriented bounding box (OBB) tree, can be used as well. Specifically, we build two sphere trees for S' and S, respectively. Each leaf of a sphere tree contains the convex hull $H(s)$ for a line segment s, and each internal node v contains a sphere that encloses the geometric objects in the children of v (Fig. 3). Clearly $H(s)$ and $H(s')$ can

intersect only if their enclosing sphere intersect. It then follows from Theorem 1 that s and s' intersect with non-zero probability only if the spheres enclosing $H(s)$ and $H(s')$ intersect. By traversing the sphere trees hierarchically, we can quickly eliminate most of the line segment pairs that have zero intersection probability and reduce the cost of checking a quadratic number of line segment pairs to a much smaller number. We omit the details of constructing sphere tree hierarchies and traversing them for collision detection, as they are well documented elsewhere (see, *e.g.*, [16]).

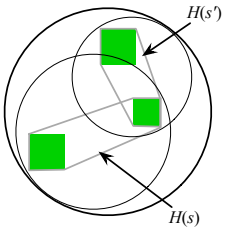


Fig. 3. A sphere tree over two uncertain line segments.

In summary, by exploiting a hierarchical representation, `FindColEvents` efficiently identifies most line segment pairs with intersection probability 0 and reduce the trivial probability bound of $[0, 1]$ to $[0, 0]$ for all of them together. For the remaining line segment pairs, which are usually small in number, their probability bounds are further refined when necessary (see next subsection).

4.4 Hierarchical Refinement of Collision Probability Bounds

We now consider the problem of refining the bounds on the intersection probability $\mathrm{P}(I_{s,s'(q)})$ for some line segment pair $s \in S$ and $s' \in S'$. To simplify the notation, we will omit the parameter q and assume that s' is translated and rotated suitably.

Computing $\mathrm{P}(I_{s,s'})$ is in essence an integration problem. Let x_1 and x_2 be the endpoints of s, and let x_3 and x_4 be the endpoints of s'. Suppose that x_i has probability density function $f_i(x_i)$ with rectangular support regions R_i. We can calculate the probability that s and s' intersect by integrating over $R_1 \times \cdots \times R_4$:

$$\mathrm{P}(I_{s,s'}) = \int_{R_1 \times \cdots \times R_4} A(x_1, \ldots, x_4)\ f_1(x_1)dx_1 \cdots f_4(x_4)dx_4, \qquad (5)$$

where $A(x_1, \ldots, x_4)$ is an index function that is 1 if and only if s and s' intersect. In two-dimensional environments, this integral is 8-dimensional.

To evaluate this integral, we decompose the integration domain $R_1 \times \cdots \times R_4$ hierarchically into a set of subdomains such that in each subdomain, the index function A is constant. By summing up the probability mass associated with all the subdomains where A is 1, we get the value for $\mathrm{P}(I_{s,s'})$. During the hierarchical decomposition process, we maintain three lists of subdomains: (i) subdomains where A is always 1, (ii) subdomains where A is always 0, and (iii) subdomains where A has mixed values (0 or 1). Interestingly, these three lists provide an upper bound and a lower bound on $\mathrm{P}(I_{s,s'})$ at any moment during the decomposition process. Let p_1 and p_2 be the probability mass associated with subdomains in list (i) and (ii), respectively. Clearly, we have $p_1 \leq \mathrm{P}(I_{s,s'}) \leq 1 - p_2$. The probability mass associated with subdomains in list (iii) is $1 - p_1 - p_2$. It represents the gap between the upper and lower bounds. To refine the bounds, we simply take a subdomain from list (iii) and decompose it further until some of the refined subdomains can be assigned to either list (i) or list (ii).

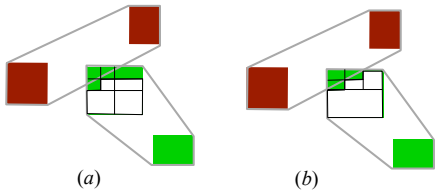


Fig. 4. Decomposing the integration domain for intersection probability calculation. (*a*) The quadtree-based procedure. (*b*) Our procedure that takes into account the geometry of intersecting line segments. The example shows that after roughly a same number of cuts, our procedure identifies a large part of the domain for which no further decomposition is needed.

To decompose an integration domain $R_1 \times \cdots \times R_4$, we take a horizontal or vertical cut on one or more of the endpoint regions R_i and obtain a set of subdomains $R'_1 \times \cdots \times R'_4$ such that R'_i is rectangular and $R'_i \subseteq R_i$ for $i = 1, 2, 3, 4$. Using Theorem 1, we can easily determine whether the index function A has constant value over a subdomain and assign the subdomain to the appropriate list.

There are various strategies to decompose a rectangular integration domain. The main goal is to assign each subdomain to list (i) or (ii) and avoid unnecessarily decomposing into a large number of tiny subdomains. One possible decomposition procedure, based on the quadtree [6], always cuts an endpoint region in the middle either horizontally or vertically (Fig. 4*a*). This is simple to implement, but does not always results in the best decomposition, as it may unnecessarily cut a domain into small pieces. Our decomposition procedure uses the geometry of the two intersecting line segments s and s' to decide where to cut. This results in better decomposition, but the trade-off is that each decomposition step is slightly more expensive. To determine how to cut, our procedure enumerates several cases that depend on the relative positions of the endpoint regions and convex hulls for s and s'. The details are not particularly important. An example decomposition is shown in Fig. 4*b* for illustration. In the experiments, our decomposition procedure usually gives slightly better performance than the quadtree-based procedure.

An alternative way of evaluating the integral in (5) is to perform Monte Carlo integration [11] by sampling from the integration domain $R_1 \times \cdots \times R_4$. Each sample consists of four points $x_1, \ldots, x_4$ with $x_i \in R_i$. Let A_i be the value of the index function A for the ith sample, and p be the value of the integral in (5). Then an estimate of p is given by

$$p_N = \frac{1}{N} \sum_{i=1}^{N} A_i. \tag{6}$$

The values $A_i, i = 1, 2, \ldots, N$ are in fact a set of independent and identically distributed (i.i.d.) random variables. Under a wide range of sampling distributions, the mean of A_i is equal to p. Let V_A be the variance of A_i. By (6), p_N is a random variable with mean p and variance V_A/N. We can then apply Chebychev's inequality and obtain

$$\mathrm{P}\left(|p_N - p)| \geq (1/\delta)(V_A/N)^{-1/2})\right) \leq \delta, \tag{7}$$

which implies that p_N converges to p at the rate $O(N^{-1/2})$. More precisely, for any δ arbitrarily small, we can determine the number of samples, N, needed to ensure that the estimate p_N does not deviate too much from p. So instead of maintaining upper and lower bounds on the collision probabilities, we can choose N large enough to get sufficiently accurate estimates for all the collision probabilities and find a minimum-cost path with high probability. Unfortunately, using Monte Carlo integration this way is not efficient (see Section 5), as it uses the same number of samples for estimation everywhere over the entire environment.

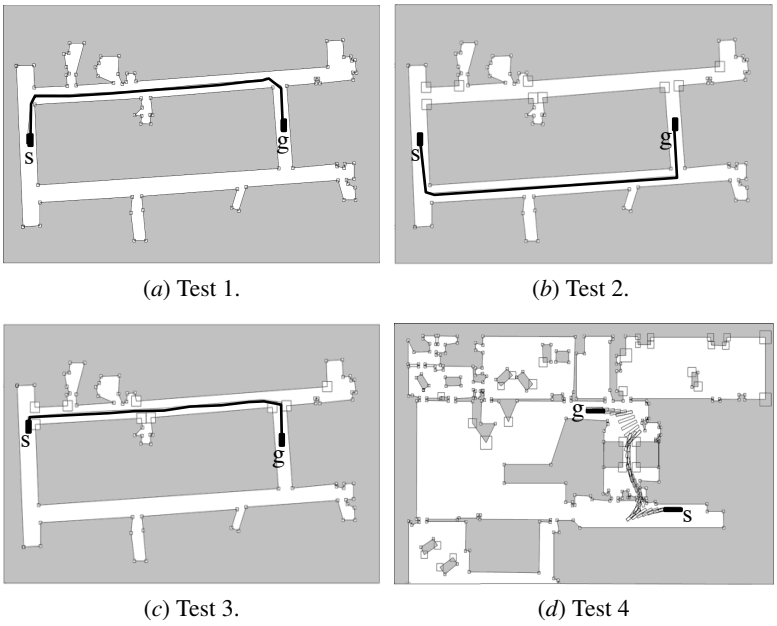
An interesting method is to combine probability bound refinement and Monte Carlo integration. We start by decomposing the integration domain as described earlier. When the probability mass associated with a subdomain is small enough, we apply the Monte Carlo method with a small number of samples to get an estimate and close the gap between the upper and lower bounds. Strictly speaking, if we do this, we cannot guarantee that the algorithm finds a minimum-cost path in U. However, if we use a sufficient number of samples for Monte Carlo integration, we can provide the guarantee with high probability. Furthermore, even when the algorithm fails to find a minimum-cost path, the cost of the resulting path is still a good approximation to the minimum cost. The reason is that due to the bound in (7), we make a mistake only when two paths have very similar cost. We use this combined method in our implementation of the algorithm, and it achieves better performance better than one that uses pure probability bound refinement.

5 Experiments

We made a preliminary implementation of our algorithm and compared it with two alternatives. One algorithm, which we developed for the purpose of comparison, is similar to BURM. It also builds an uncertainty roadmap. However, instead of refining the probability bounds incrementally when necessary, it estimates the exact probabilities using Monte Carlo integration. We call this algorithm MCURM. In our tests, MCURM uses 100 samples to evaluate each intersection probability $\mathrm{P}(I_{s,s(q)})$. The other algorithm that we compared is Lazy-PRM [2], which does not take into account uncertainty during planning.

In our tests, all three algorithms use the same sampling strategy, which is a hybrid strategy consisting of the bridge test and the uniform sampler [7]. We ran the algorithms on each test case and repeated 30 times independently. The performance statistics reported here are the averages of 30 runs. In each run, the three algorithms used the same set of sampled configurations. So the performance difference results from the way they find a minimum-cost path in an (uncertainty) roadmap rather than random variations in sampling the configuration space.

The test results are shown in Fig. 5 and Table 1. In tests 1–3, the robot has a rectangular shape and only translates. In these three tests, the environments are similar. The robot essentially chooses between two corridors to go from the start to the goal position. The main differences among the tests are (i) the level of uncertainty in the obstacle geometry and (ii) the robot start position. In test 1 (Fig. 5*a*), the uncertainty is low and roughly the same everywhere. So the robot chooses the upper corridor,



(*a*) Test 1. (*b*) Test 2.

(*c*) Test 3. (*d*) Test 4

Fig. 5. Test environments and results. The boxes around the vertices mark the support regions of the probability distributions modeling the endpoint positions of line segments forming the obstacle boundaries.

based mainly on the path length consideration. However, it is interesting to observe that although a shortest path with respect to the path length normally touches obstacle boundaries, our minimal-cost path stays roughly in the middle of the corridor. It does so to avoid collision due to the uncertainty in obstacle geometry. In test 2, the upper corridor has substantially higher uncertainty than the lower corridor. On balance, it is better for the robot to choose the slightly longer, but safer lower corridor (Fig. 5*b*). In test 3, the uncertainty in obstacle geometry remains the same as that in test 2, but the start position for the robot moves higher. The robot again decides to go through the upper corridor, because despite the higher collision risk of the upper corridor, it is much shorter than the lower corridor (Fig. 5*c*).

In test 4, the robot can both translate and rotate. To reach its goal, the robot can either take the risk of collision and squeeze through the narrow passage or make a long detour. It is not obvious which choice is better. The answer depends, of course, on the cost of collision. In this case, the robot decides to take the riskier, but shorter path (Fig. 5*d*). Interestingly, our algorithm finds two paths of similar cost, depending on the set of sampled configurations. One path veers to the right (Fig. 5*d*) when it approaches the obstacle near the lower entrance to the narrow passage, and the other veers to the left. This is in fact not surprising, because regardless of whether the path veers to the the left or right, the uncertainty that the path encounters remains similar and the path length does not differ by much.

Table 1. Performance statistics.

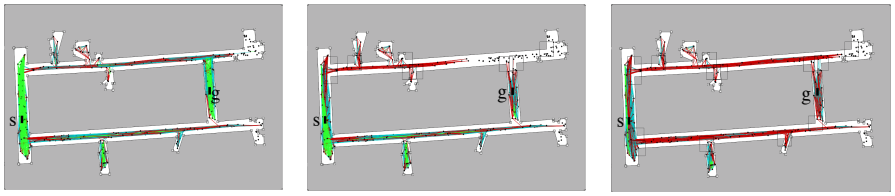
Test Env.	No. Nodes	Cost			Time (s)		
		BURM	MCURM	Lazy-PRM	BURM	MCURM	Lazy-PRM
1	300	699	699	789	59	3,119	42
2	300	741	741	1,059	72	2,871	42
3	300	761	760	891	74	2,994	35
4	500	526	526	668	61	2,534	36

Now let us look at the performance statistics. For each test case, Table 1 lists the number of nodes in the (uncertainty) roadmap, the running times, and the cost of the paths found by the three algorithms. Note that the absolute running times reported in Table 1 are a little slow, as our implementation is still preliminary and does not optimize the speed of important primitive geometric operations such as the intersection test. We plan to improve the implementation in the future. However, this does not significantly affect the comparison of the three algorithms based on their relative performance, as they use the same implementation of primitive operations.

BURM and MCURM find paths with almost the same cost. BURM is, however, 40–50 times faster. This clearly demonstrates the advantage of the approach of evaluating uncertainty hierarchically at multiple resolutions.

The comparison between BURM and Lazy-PRM is even more interesting. As expected, BURM finds paths with lower cost, as it takes uncertainty into account during planning. However, it is somewhat surprising that BURM is not much slower than Lazy-PRM: BURM is only 2 times slower than Lazy-PRM, while it is at least 40 faster than MCURM. The reason is that uncertainty comes into play in deciding the best path when the robot operates in close proximity of the obstacles. This usually happens in localized regions of the configuration space only. BURM takes advantage of this by maintaining bounds on collision probabilities rather than calculating the exact probabilities. It refines these bounds incrementally by exploiting bounding volume hierarchies on the geometry of the obstacles and the robot and by hierarchically decomposing the integration domain for collision probability calculation. The running time comparison with Lazy-PRM provides further evidence on the advantage of our approach.

To better understand the behavior of BURM, we applied it to an environment similar to that in tests 1–3 and varied the uncertainty level in the obstacle geometry. The resulting bounded uncertainty roadmaps are shown in Fig. 6. The edges of the roadmaps are colored to indicate how tight the associated collision probability bounds are. The roadmap in Fig. 6*a* serves as a reference point for comparison. The uncertainty is low in both the upper and lower corridors, and the collision probability bounds are refined to various degrees. As the uncertainty gets higher in the upper corridor, the collision probability bounds there are tightened to differentiate the quality of the paths (Fig. 6*b*). It is also interesting to observe that the upper corridor is not explored as much, because the paths in the lower corridor are far better. Finally, as the uncertainty in the lower corridor also increases, both corridors must be explored,



(*a*) (*b*) (*c*)

Fig. 6. Color-coded BURMs. Red, gray, and blue-green marks edges with tight, intermediate, and loose collision probability bounds, respectively. Bright green marks edges with collision probability 0.

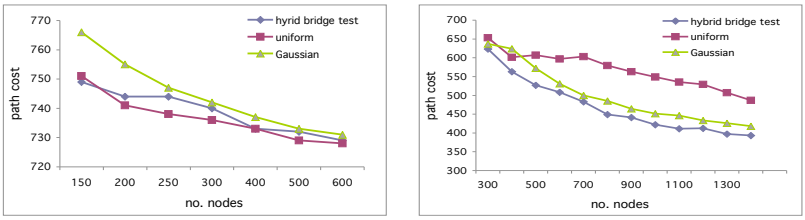
and the collision probability bounds carefully tightened in order to determine the best path (Fig. 6*c*).

Bounded uncertainty roadmaps can be used with any existing sampling strategies. To demonstrate this, we performed additional tests by varying the sampling strategy used and the number of nodes in the uncertainty roadmap. We tried two additional strategies: the uniform sampler and the Gaussian sampler [3]. For all sampling strategies, as the roadmap size increases, the running time increases correspondingly. Two plots of representative results are shown in Fig. 7. They indicate that the cost of the minimum-cost path found decreases with the roadmap size up to a certain point and then stabilizes. So one way of minimizing the path cost is to run our algorithm in an "anytime" fashion by gradually adding more nodes to the roadmap. The effect of different sampling strategies is more pronounced in complex environments. In the simpler environment (see Fig. 5*b*), when the number of roadmap nodes is sufficiently large, the results obtained by the different sampling strategies are comparable. When the number of nodes is small, the Gaussian sampler does not behave very well, as it biases sampling towards the obstacle boundaries, resulting in high collision cost. In the more complex environment (see Fig. 5*d*), which contains several narrow passages, the hybrid bridge test and the Gaussian sampler have clear advantages over the uniform sampler. Just as in classic motion planning, effective sampling strategies for constructing uncertainty roadmaps are important and require further investigation.

6 Conclusion

We have introduced the notion of a bounded uncertainty roadmap and used it to extend sampling-based algorithms for planning under uncertainty in environment maps. By evaluating uncertainty hierarchically at multiple resolutions in different regions of a robot's configuration space, our approach greatly improves planning efficiency. Experimental results, based on a preliminary implementation of our planning algorithm, demonstrate that it is highly effective.

There are many interesting directions for future work. The main idea of our approach, evaluating uncertainty hierarchically at multiple resolutions, is not restricted to the particular path cost function used here. For example, our current path cost function sums up the collision costs for the configurations along a path. Sometimes



(*a*) (*b*)

Fig. 7. The change in the cost of the minimum-cost path found as a function of the sampling strategy and the number of nodes in the uncertainty roadmap.

it may be more suitable to take the maximum of rather than sum up the collision costs. Our idea can be applied to this new path cost function with small modifications. We would also like to understand the effect of sampling strategies on the running time and the quality of the result for our algorithm with respect to particular classes of path cost functions. One idea is to sample a robot's configuration space adaptively and adjust the sampling distribution based on the collision probabilities of previously sampled configurations. Finally, we will implement our algorithm and test it in three-dimensional environments.

Acknowledgments

This work is supported in part by MoE AcRF grant R-252-000-327-112 and NSF grants CCF-0634803 and FRG-0354543.

References

1. R. Alterovitz, T. T. Siméon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty. In *Proc. Robotics: Science and Systems*, 2007.
2. R. Bohlin and L.E. Kavraki. Path planning using lazy PRM. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 521–528, 2000.
3. V. Boor, M.H. Overmars, and F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 1018–1023, 1999.
4. B. Burns and O. Brock. Sampling-based motion planning with sensing uncertainty. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 3313–3318, 2007.
5. H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of Robot Motion : Theory, Algorithms, and Implementations*, chapter 7. The MIT Press, 2005.
6. M. de Berg, M. van Kreveld, M.H. Overmars, and O. Schwarzkopf. *Computaional Geometry: Algorithms and Applications*. Springer, 2000.
7. D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 4420–4426, 2003.
8. D. Hsu, J.C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Int. J. Robotics Research*, 25(7):627–643, 2006.

9. D. Hsu, J.C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 2719–2726, 1997.
10. L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.
11. M.H. Kalos and P.A. Whitlock. *Monte Carlo Methods*, volume 1. John Wiley & Sons, New York, 1986.
12. H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems*, 2008.
13. J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.
14. S.M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
15. S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 473–479, 1999.
16. M. Lin and D. Manocha. Collision and proximity queries. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 35. CRC Press, 2004.
17. P. Missiuro and N. Roy. Adapting probabilistic roadmaps to handle uncertain maps. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 1261–1267, 2006.
18. C. Papadimitriou and J.N. Tsisiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.
19. J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. Int. Jnt. Conf. on Artificial Intelligence*, pages 477–484, 2003.
20. R.D. Smallwood and E.J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.
21. T. Smith and R. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. Uncertainty in Artificial Intelligence*, 2005.
22. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005.
23. Y. Yu and K. Gupta. Sensor-based roadmaps for motion planning for articulated robots in unknown environments: Some experiments with an eye-in-hand system. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 1070–1714, 1999.

A Proof of Theorem 1

Proof. To prove part (1), observe that since $H(s) \cap H(s') = \emptyset$, line segments s and s' has no intersection for all pairs s and s' whose endpoints lie in $R(s)$ and $R(s')$, respectively. This immediately implies that the intersection probability is 0.

Now consider part (2). We start with a simple observation. Let σ be a fixed line segment such that the endpoints of σ lie outside $H(\sigma')$ for some line segment σ' with uncertain endpoint positions and σ does not intersect with $R(\sigma')$. Then either σ intersects with σ' for *all* σ' whose endpoints lie in $R(\sigma')$, or σ intersects with *none* of them. Since $R(s) \cap H(s') = \emptyset$, the endpoints of s must lie outside of $H(s')$. Furthermore, s does not intersect with $R(s')$, as $R(s') \cap H(s) = \emptyset$. From our observation, it follows that s intersects with all s' with endpoints in $R(s')$ or intersects with none of them at all, and this is true for all s with endpoints lying in $R(s)$. Now, since $H(s) \cap H(s') \neq \emptyset$, there exists at least one pair of intersecting line segments s and s' with endpoints in $R(s)$ and $R(s')$. By the convexity of $R(s), R(s'), H(s)$ and $H(s')$, we conclude that all line segment pairs s and s' intersect. Thus the probability of intersection is 1.