Asymptotically Near-Optimal Planning With Probabilistic Roadmap Spanners

James D. Marble, *Member, IEEE*, and Kostas E. Bekris, *Member, IEEE*

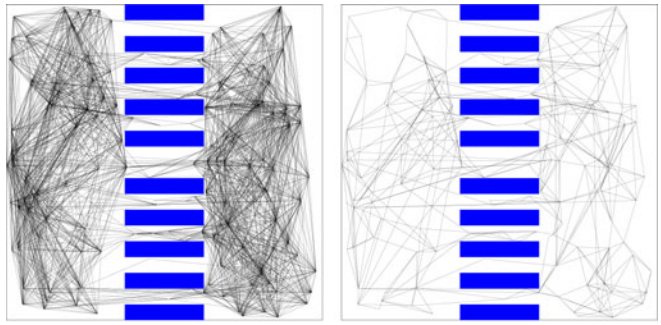
***Abstract*—Asymptotically optimal motion planners guarantee that solutions approach optimal as more iterations are performed. A recently proposed roadmap-based method, i.e., the PRM* approach, provides this desirable property and minimizes the computational cost of generating the roadmap. Even for this method, however, the roadmap can be slow to construct and quickly grows too large for storage or fast online query resolution, especially for relatively high-dimensional instances. In graph theory, there are algorithms that produce sparse subgraphs, which are known as graph spanners, that guarantee near-optimal paths. This paper proposes different alternatives for interleaving graph spanners with the asymptotically optimal PRM* algorithm. The first alternative follows a sequential approach, where a graph spanner algorithm is applied to the output roadmap of PRM*. The second one is an incremental method, where certain edges are not considered during the construction of the roadmap as they are not necessary for a roadmap spanner. The result in both cases is an asymptotically *near*-optimal motion planning solution. Theoretical analysis and experiments performed on typical, geometric motion planning instances show that large reductions in construction time, roadmap density, and online query resolution time can be achieved with a small sacrifice of path quality through roadmap spanners.**

***Index Terms*—Motion planning, near-optimality, probabilistic roadmaps.**

I. Introduction

SAMPLING-BASED algorithms correspond to a practical framework to solve high-dimensional motion planning problems. The probabilistic roadmap method (PRM) [1] is an instance of such a method that uses an offline phase to construct a graph that represents the structure of the configuration space (C-space). This graph, which is known as a roadmap, can be queried in an online phase to quickly provide solutions.

Traditionally, PRM and many related variants [2]–[7] focused on quickly finding a feasible solution. Pursuing only this objective may result in reduced solution quality, which can be evaluated given a variety of measures depending on the underlying task. For instance, clearance from obstacles or smoothness can be used to evaluate the quality of the solution path [8]. This study focuses on popular path quality measures that are metric functions, such as path length or traversal time. One standard way to improve solution quality is through path smoothing, a postprocessing step applied to the path returned by a planner. A more sophisticated way to improve path quality is through hybridization graphs, which combine multiple solutions into a higher quality one that uses the best portions of each input path [9]. There are also algorithms that attempt to produce roadmaps, which return paths that are deformable to optimal ones after the application of smoothing [10], [11]. These smoothing-based techniques, however, can be expensive for the online resolution of a query, especially when multiple queries must be answered given the same input roadmap.



(a) Dense roadmap (b) Sparse spanner

Fig. 1. (a) Roadmap constructed by an algorithm that is guaranteed to converge to optimal solution costs and (b) another roadmap constructed in such a way that guarantees convergence to *near*-optimal solution costs but results in far fewer edges.

Alternatively, it is possible to construct larger, denser roadmaps that better sample the C-space by investing more preprocessing time. For instance, a planner that attempts to connect a new sample to every existing node in the roadmap will converge to solutions with optimal costs, a property known as asymptotic optimality. While roadmaps with this property are desirable for their high path quality, their large size can be problematic. Large roadmaps impose significant costs during construction, storage, transmission, and online query resolution; therefore, they may not be appropriate for many applications. The recently proposed k-PRM* algorithm [12] minimizes the number k of neighbors each new roadmap sample has to be tested for connection while still providing asymptotic optimality guarantees. Even so, the density of roadmaps produced by k-PRM* can still be high, resulting in slow online query resolution times.

Manuscript received December 30, 2011; revised August 14, 2012; accepted December 8, 2012. Date of publication January 11, 2013; date of current version April 1, 2013. This paper was recommended for publication by Associate Editor T. Simeon and Editor D. Fox upon evaluation of the reviewers' comments. This work was supported by the National Science Foundation under Award CNS 0932423. Any opinions and findings expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsor.

J. D. Marble was with the Department of Computer Science and Engineering, University of Nevada, Reno, NV 89557 USA. He is now with Sierra Nevada Corporation, Sparks, NV 89434 USA (e-mail: midorikid@gmail.com).

K. E. Bekris was with the Department of Computer Science and Engineering, University of Nevada, Reno, NV 89557 USA. He is now with the Department of Computer Science, Rutgers University, Piscataway, NJ 08854 USA (e-mail: kostas.bekris@cs.rutgers.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2012.2234312

This paper argues that a viable alternative is to construct roadmaps with asymptotic near-optimality guarantees. By relaxing the optimality requirement, it is possible to construct roadmaps that are sparser, faster to build, and can answer queries faster, while providing solution paths that are arbitrarily close to optimal. The idea is illustrated in Fig. 1.

The theoretic foundations for this study lie in graph theory. In particular, graph spanners are sparse subgraphs with guarantees on path quality. Spanner construction methods filter edges of the original graph and guarantee that any shortest path between nodes in the sparse subgraph has cost at most $t \times c^*$, where c^* is the optimum path cost in the original graph, and t is an input parameter called the stretch factor. An existing method in the motion planning literature, which created roadmaps that contained Useful Cycles [13] was in fact producing graph spanners. In this framework, roadmap edges that pass the "usefulness" test are added to the roadmap because not doing so would violate path quality guarantees, similar to those of graph spanners. Nevertheless, since this method did not start from a roadmap with optimality guarantees, it did not provide any properties related to optimality.

In this study, graph spanner algorithms are combined with asymptotically optimal roadmap planners to produce planners that construct roadmap spanners. These are roadmap methods with asymptotic near-optimality guarantees. Two algorithms are proposed in this paper.

1) The first method builds a roadmap spanner in a sequential manner given an asymptotically optimal roadmap.
2) A second proposed alternative interleaves roadmap construction with a spanner preserving filter to directly construct a roadmap spanner in an efficient way.

The resulting roadmaps are sparse, which is beneficial in any application where the density of the roadmap matters.

A. Related Work

Computational efficiency: There has been a plethora of techniques on how to sample and connect configurations so as to achieve computational efficiency in roadmap construction via a sampling-based process [2]–[6], [14]. Certain algorithms, such as the visibility-based Roadmap [4], the Incremental Map Generation algorithm [6], and the Reachability Roadmap Method [15], focus on returning a connected roadmap that covers the entire configuration space. A reachability analysis of roadmap techniques suggested that connecting roadmaps is more difficult than covering the C-space [7]. Furthermore, a method is available to characterize the contribution of a sample to the exploration of the C-space [16], but it does not address how the connectivity between samples contributes to the resulting path quality once the space has been explored.

Path Quality: Work on creating roadmaps that contain high-quality paths has been motivated by the objective to efficiently resolve queries without the need for a postprocessing optimization step of the returned path. One technique aims to compute all different homotopic solutions [11], while Path Deformation Roadmaps compute paths that are deformable to optimal ones [10]. Another approach inspired by Dijkstra's algorithm extracts optimal paths from roadmaps for specific queries [17] but may require very dense roadmaps. The Useful Cycles approach implicitly creates a roadmap spanner with small number of edges [13] and has been combined with the Reachability Roadmap Method to construct connected roadmaps that cover 2-D and 3-D C-spaces to provide high-quality paths [18]. A method has been introduced that filters nodes from a roadmap if they do not improve path quality measures [19].

Tree-based planners and dynamics: Roadmaps cannot be constructed for problems where there is no solution for the two-point boundary value problem, i.e., it is not easy to compute a path that connects exactly two states of a moving system. On the other hand, tree-based kinodynamic planners [20], [21] can operate on such environments. Tree-based algorithms are very effective for single-query planning. They do not provide, however, the preprocessing properties of roadmaps, and they tend to be slower for multiquery applications. Regarding sparsity, a tree is already a sparse graph and it becomes disconnected when removing edges. It has been shown that the RRT algorithm produces arbitrarily bad paths with high probability and will miss high-quality paths even if they are easy to find [12], [22]. Anytime RRT [23] has been proposed as an approach that practically improves path quality in an incremental manner. Tree-based planners for kinodynamic problems can still benefit from an approximate roadmap to estimate distances between states and the goal region that take into account C-space obstacles. Such distance estimates can be used as a heuristic in tree expansion to bias the selection of states closer to the goal and solve problems with dynamics faster [24], [25].

Asymptotic Optimality: The RRG, RRT*, and PRM* families of algorithms [12] provide asymptotic optimality for general configuration spaces. RRG and RRT* are based on RRT, which is a tree-based planner. The anytime RRT* approach [26] extends RRT* with anytime planning in dynamic environments and can incrementally improve path quality. PRM* is a modification of standard PRM. The proposed techniques in this paper are based on k-PRM*, i.e., a variation of PRM*, which will be described in detail in Section II-A.

B. Contribution

This paper describes two methods to construct roadmap spanners based on previous work by the authors [27], [28]. It is shown that both methods guarantee a constant factor of asymptotic optimality, i.e., provide asymptotic near-optimality. The first alternative is the *Sequential Roadmap Spanner* (SRS) method [27], which has the following characteristics.

1) It constructs a sparse graph given a dense roadmap.
2) It provides asymptotic guarantees on the number of edges.
3) It provides asymptotically near-optimal solutions.
4) It has the same asymptotic time complexity as PRM.

SRS removes edges from a roadmap with path quality guarantees, while retaining near-optimal path quality utilizing state-of-the-art graph spanner algorithms with good complexity performance. In this manner, online query resolution time is shortened, while simultaneously reducing the space required for storing and transmitting the roadmap.

The second alternative is the *Incremental Roadmap Spanner* (IRS) method [28], which has the following characteristics.

1) It directly constructs sparse roadmaps using a sampling and filtering process.
2) In environments where collision detection is expensive, it constructs roadmaps in a computationally efficient way.
3) It provides asymptotically near-optimal solutions.
4) It has an asymptotic time complexity close to that of PRM.

IRS can incrementally construct a roadmap spanner in a continuous space, while most graph spanner algorithms are formulated to operate on an existing roadmap. Because IRS employs the framework of an asymptotically optimal planner, i.e., PRM* [12], it provides asymptotic near-optimality guarantees that related approaches, such as the Useful Cycles method, do not [13]. IRS is shown to produce significantly sparser roadmaps than k-PRM* experimentally.

Both methods balance two extremes in terms of motion planning solutions. On one hand, the connected component heuristic for PRM can connect the space very quickly with a very sparse roadmap, but can produce solutions with arbitrarily high cost. On the other hand, the k-PRM* algorithm provides asymptotically optimal roadmaps that tend to be very dense and slow to construct. With IRS, it is possible to tune the solution quality degradation relative to k-PRM* and select a parameter, i.e., the stretch factor, that will return solutions arbitrarily close to the optimal ones while still constructing sparse roadmaps efficiently.

The theoretical guarantees on path quality that these techniques provide are tested empirically in a variety of motion planning problems in $SE(2)$ and $SE(3)$. Experiments show that as the desired stretch factor increases, most of the roadmap edges can be removed while increasing mean path length by a relatively small amount. Often, this increase in path length can be significantly reduced by utilizing smoothing. Path degradation is most pronounced for paths that are very short, while longer paths are less affected. The sparsity of the roadmaps produced is a valuable feature in itself, but with IRS, a marked decrease in construction time is also measured.

Relative to the authors' previous work [27], [28], this paper provides a more detailed description and analysis of the two proposed algorithms under a common framework for the development of roadmap spanners. Furthermore, it provides a direct comparison of the two methods using a new, common set of experiments. The experiments suggest that the incremental approach is able to produce sparser roadmaps than the sequential alternative for the same stretch factor.

II. Foundations

A robot can be abstracted as a point in a d-dimensional configuration-space (C-space) where the set of collision-free configurations is defined as $C_{\text{free}} \subset C$ [29]. The experiments performed in this paper take place in the spaces of 2-D and 3-D rigid body transformations ($SE(2)$ and $SE(3)$ correspondingly). The proposed methods are also applicable to any C-space that is a metric and probability space for reasons described in Section III. Once C_{free} can be calculated for a particular robot, one needs to specify initial and goal configurations to define an instance of the path planning problem:

Definition 1 (Path Planning Problem): Given a set of collision-free configurations $C_{\text{free}} \subset C$, initial and goal configurations $q_{\text{init}}, q_{\text{goal}} \in C_{\text{free}}$ find a continuous curve $\sigma \in \Sigma = \{\rho | \rho : [0,1] \rightarrow C_{\text{free}}\}$, where $\sigma(0) = q_{\text{init}}$ and $\sigma(1) = q_{\text{goal}}$.

The PRM algorithm [1] can find solutions to the path planning problem by first sampling configurations in C_{free} and then trying to connect them to neighboring configurations with local paths. It starts with an empty roadmap and then iterates until some stopping criterion is satisfied. For each iteration, a configuration in C_{free} is sampled and a set of neighbors is identified. For each of these neighbors, an attempt is made to connect them to the sampled configuration with a local planner. If such a curve exists in C_{free}, i.e., it is collision free, an edge between the two configurations is added to the roadmap.

The offline phase of the algorithm is completed once some user-defined stopping criterion is met. The result is a graph $G = (V, E)$ that reflects the connectivity of C_{free} and can be used to answer query pairs of the form $(q_{\text{init}}, q_{\text{goal}}) \in C_{\text{free}} \times C_{\text{free}}$, where q_{init} is the starting configuration, and q_{goal} is the goal configuration. This type of graph is known as a roadmap. The procedure for querying it is to add q_{init} and q_{goal} to the roadmap in the same way sampled configurations are added during the offline phase. Then, a discrete graph search is performed to find a path on the roadmap between the two configurations.

A. k-PRM*

Asymptotic optimality is the property that, given enough time, solutions to the path planning problem will almost surely converge to the optimum, as defined by a cost function.

Definition 2 (Asymptotic Optimality in Path Planning): An algorithm is *asymptotically optimal* if, for any path planning problem $(C_{\text{free}}, q_{\text{init}}, q_{\text{goal}})$ and cost function $c : \Sigma \rightarrow \mathbb{R}_{\geq 0}$ with a robust optimal solution of finite cost c^*, the probability of finding a solution of cost c^* converges to 1 as the number of iterations approaches infinity.

Asymptotic optimality is defined only for *robustly feasible* problems, as defined in the presentation of k-PRM* [12]. Such problems have a minimum clearance around the optimal solution and cost functions with a continuity property.

The method with which neighbors are selected is an important variable in the PRM. In *fully connected* PRM, all existing samples in the roadmap are considered neighbors, regardless of distance. This results in a highly dense roadmap. The number of edges can be reduced by restricting the maximum distance that two samples are considered neighbors, as in *simple* PRM. The number of edges still grows quadratically with the number of samples in this case. The density of the roadmap can be further reduced by considering only a fixed number of the nearest samples as neighbors, as in *k-nearest* PRM. Although k-nearest PRM produces the sparsest roadmaps of the three, it does not provide asymptotic optimality [12]. The properties of these and the following variations are described in Table I.

The PRM* and k-PRM* algorithms rectify this by selecting a number of neighbors, which is a logarithmic function

TABLE I
VARIATIONS OF PRM AND THEIR PROPERTIES

algorithm	edges	optimal?
complete PRM	$O(n^2)$	asymptotically
simple PRM	$O(n^2)$	asymptotically
k-nearest PRM	$O(kn)$	no
PRM*	$O(n \log n)$	asymptotically
k−PRM*	$O(n \log n)$	asymptotically
SRS	$O(an^{1+\frac{1}{a}})$	asymptotically near
IRS	$O(n \log n)$	asymptotically near

Algorithm 1 GRAPHSPANNER(V, E, t)

1: sort E by non-decreasing weight
2: $E_S \leftarrow \emptyset, G_S \leftarrow (V, E_S)$
3: **for all** $(v, u) \in E$ **do**
4: **if** SHORTESTPATH$(V, E_S, v, u) > t \cdot$WEIGHT(v, u) **then**
5: $E_S \leftarrow E_S \cup \{(v, u)\}$
6: **end if**
7: **end for**
8: **return** G_S

of the current size of the roadmap. Specifically, for k-PRM*, the number of nearest neighbors is $k(n) = k_{\text{PRM}} \log n$, where $k_{\text{PRM}} > e(1 + 1/d)$, n is the number of roadmap nodes, and d is the dimensionality of the space. Roadmaps that are constructed by these variations are asymptotically optimal [12]. Nevertheless, the number of neighbors selected for connection attempts still grows with each iteration.

B. Graph Spanners

A graph spanner, as formalized in the related literature [30], is a sparse subgraph. Given a weighted graph $G = (V, E)$, a subgraph $G_S = (V, E_S \subset E)$ is a t-spanner of G if for all pairs of vertices $(v_1, v_2) \in V$, the shortest path between v_1 and v_2 in G_S is no longer than t times the shortest path between v_1 and v_2 in G. Because t specifies the amount of additional length allowed in the solution returned by the spanner, it is known as the *stretch factor* of the spanner.

A simple method for spanner construction, which is a generalization of Kruskal's algorithm for the minimum spanning tree, is given in Algorithm 1 [31]. Instead of accepting only edges that connect disconnected components, this algorithm also accepts edges that add useful cycles. Kruskal's algorithm is recovered by setting t to a large value, such that no cycle is useful enough to be added.

The inclusion criteria on line 4 ensure that no edges required to maintain the spanner property are left out. From the global ordering of the edges of line 1, it has been shown that the number of edges retained is reduced from a potential $O(n^2)$ to $O(n)$ [31].

The quadratic number of shortest path queries puts the time complexity into $O(n^3 \log n)$; however, much work has been done to find algorithms that reduce this. Most perform clustering in a preprocessing step, and one method uses this idea to reduce the time complexity to $O(m)$, where m is the number of edges [32]. It is this algorithm that was chosen for the sequential approach presented in this study.

TABLE II
SPANNER ALGORITHMS AND THEIR PROPERTIES

algorithm	stretch	edges	time
Althöfer et al. [31]	$2a - 1$	$O(n^{1+\frac{1}{a}})$	$O(an^{1+\frac{1}{a}})$
Cohen [35]	$2a + \epsilon$	$O(an^{1+\frac{1}{a}})$	$O(an^{\frac{1}{a}})$
Awerbuch et al. [36]	$64a$	$O(an^{1+\frac{1}{a}})$	$O(mn^{\frac{1}{a}})$
Roditty et al. [37]	$2a - 1$	$O(n^{1+\frac{1}{a}})$	$O(an^{2+\frac{1}{a}})$
Thorup et al. [38]	$2a - 1$	$O(an^{1+\frac{1}{a}})$	$O(amn^{\frac{1}{a}})$
Baswana et al. [32]	$2a - 1$	$O(an^{1+\frac{1}{a}})$	$O(am)$

Algorithm 2 SEQUENTIALROADMAPSPANNER(a)

1: $V \leftarrow \emptyset; E \leftarrow \emptyset$
2: k−PRM*(V, E)
3: RANDOMIZEDSPANNER(V, E, a)
4: **return** (V, E)

A number of alternatives take advantage of the implicit weights of a Euclidean metric space to improve performance. Many of these operate on *complete* Euclidean graphs [33], [34]. They cannot be easily used in C-spaces with obstacles because obstacles remove edges of the complete graph. Table II lists a range of spanner algorithms that can be applied to weighted graphs along with their properties.

III. APPROACH

The central concept proposed in this study is that of asymptotic *near*-optimality. This new property is a relaxation of asymptotic optimality that permits an algorithm to converge to a solution that is within t times the cost of the optimum.

Definition 3 (Asymptotically Near-Optimal Path Planning): An algorithm is *asymptotically near-optimal* if, for any path planning problem $(C_{\text{free}}, q_{\text{init}}, q_{\text{goal}})$ and cost function $c : \Sigma \rightarrow \mathbb{R}_{\geq 0}$ that admit a robust optimal solution with finite cost c^*, the probability that it will find a solution of cost $c \leq tc^*$ for some stretch factor $t \geq 1$ converges to 1 as the number of iterations approaches infinity.

The high-level approach is to combine the construction of an asymptotically optimal roadmap with the execution of a spanner algorithm. These two tasks can be performed either sequentially (first find the roadmap and then compute the spanner), or incrementally by interleaving the steps of each task.

A. Sequential Roadmap Spanner

An asymptotically optimal roadmap G generated by k-PRM* can be used to construct an asymptotically *near*-optimal roadmap G_S by selectively removing existing edges. Applying an appropriate spanner algorithm to the roadmap accomplishes this and is referred to in this paper as the SRS algorithm. Algorithm 2 outlines the operation of the SRS method at a high level. It first makes a call to the k-PRM* algorithm, which populates the graph (V, E), and then SRS computes the spanner of (V, E). A stopping criterion is assumed to be used by k-PRM* so that it returns in finite time.

While any number of spanner algorithms could be used for this purpose, here, a method with linear time complexity with

Algorithm 3 RANDOMIZEDSPANNER(V, E, a)

1: $E_S \leftarrow \emptyset$
2: $K_0 \leftarrow \big\{\{v\} | v \in V\big\}$
3: **for** $i \in \{1, \ldots, \lfloor \frac{a}{2} \rfloor\}$ **do**
4: $\quad K_i \leftarrow \emptyset$
5: $\quad$ **for** $c \in K_{i-1}$ **do**
6: $\quad\quad$ **if** UNIFORMRANDOM$(0,1) < \|V\|^{-\frac{1}{a}}$ **then**
7: $\quad\quad\quad K_i \leftarrow K_i \cup \{c\}$
8: $\quad\quad\quad V_i \leftarrow V_i \cup c$
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: $\quad$ **for** $v \in V - V_i$ **do**
12: $\quad\quad n_v \leftarrow$ nearest cluster in K_i to v
13: $\quad$ **end for**
14: $\quad$ ADDSPANNEREDGES(V, V_i, E, E_S, K, n)
15: $\quad$ **for** $e \in E$ **do**
16: $\quad\quad (v, u) \leftarrow e$
17: $\quad\quad$ **if** v and u are in the same cluster of K_i **then**
18: $\quad\quad\quad E \leftarrow E - \{e\}$
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: **end for**
22: CLUSTERJOINING(K, E, a, E_S)
23: **return** E_S

respect to the number of edges is adopted. The graph spanner method employed by SRS is outlined below.

1) Randomized $(2a-1)$ Spanner Algorithm [32]: The input to this approach (see Algorithm 3) corresponds to the sets of vertices and weighted edges of the roadmap, V and E, and the spanner parameter a. The stretch factor t is a function of the input parameter a: $t = (2a-1)$. The output is E_S, i.e., the edges of the spanner. There are two parts in the algorithm. First, clusters are formed. Second, the clusters are joined to each other.

A cluster c is a set of vertices. A clustering K_i is the set of clusters at iteration i. Vertices only belong to one cluster in any clustering. Before the first iteration, each vertex is made into its own, singleton cluster to form the initial K_0 clustering. With each successive iteration, the following steps are performed.

1) A subset of the previous iteration's clusters is sampled randomly [see Fig. 2(b)].
2) The vertices in the remaining, unsampled clusters are then split up and added to their neighboring clusters [see Fig. 2(c)].
3) Vertices with no adjacent clusters in the *current* clustering are connected to all adjacent clusters from the *previous* clustering [see Fig. 2(d)].

This is repeated until $\lfloor \frac{a}{2} \rfloor$ iterations have been performed. In the second part of the algorithm (line 22), the clusters are joined by the shortest edge that connects them to their neighbors [see Fig. 2(e)].

Two auxiliary functions will be useful in the definition of the $(2a-1)$ spanner algorithm. The function $E(v, c)$ returns all edges in the set E from the vertex v to vertices in cluster c. Similarly, the function $E(c, c')$ returns all edges in E connecting any vertex in cluster c to any in cluster c'.

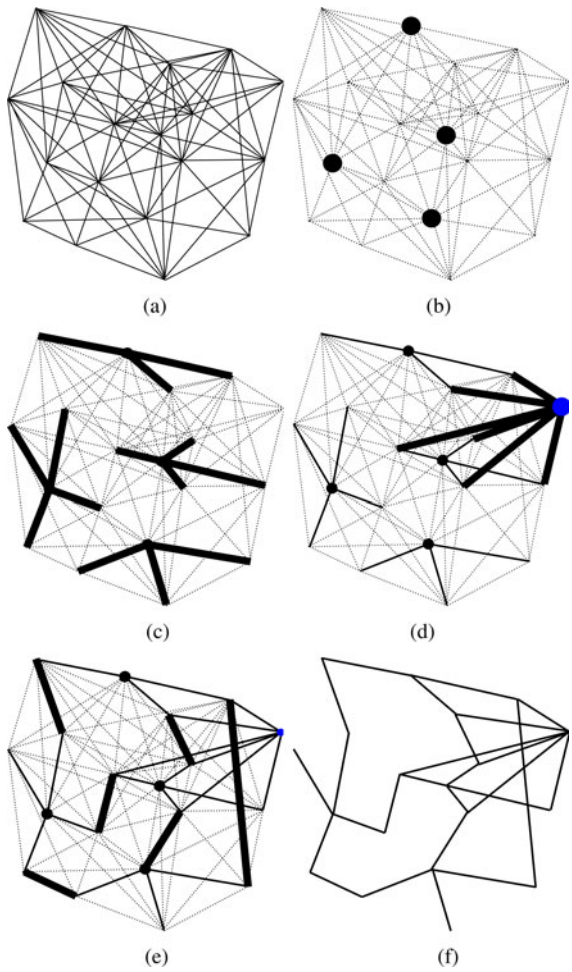


Fig. 2. Randomized $(2a-1)$ spanner algorithm for $a = 2$. (a) Initial roadmap. (b) Sample clusters. (c) Closest cluster. (d) No adjacent cluster. (e) Cluster joining. (f) Final spanner.

The clustering K_0 is initialized (line 2) so that each vertex becomes its own cluster. Lines 5–7 randomly sample the last iteration's clusters with probability $n^{-\frac{1}{a}}$ [see Fig. 2(b)]. A probability smaller than 1 has the important effect of reducing the number of clusters at each iteration. As a notational convenience, V_i represents the vertices that belong to sampled clusters at iteration i (line 8).

Sampled clusters are expanded in lines 11–14. Here, the closest, neighboring, sampled cluster, n_v, is found for each vertex that is not a member of a sampled cluster. The algorithm shown in Algorithm 4 adds edges to the spanner that connect the new vertices to their clusters.

For those vertices that are adjacent to sampled clusters, lines 3–5 add the shortest edge to the nearest sampled cluster and discards the others. To maintain the spanner property, edges to other unsampled clusters that are shorter than this must also be retained (lines 6–12). At the conclusion of each iteration (see lines 15–20 in Algorithm 3), intracluster edges are removed from E.

After all $\lfloor \frac{a}{2} \rfloor$ iterations have completed, $K_{\lfloor \frac{a}{2} \rfloor}$ contains the final clusters, each of which is a tree rooted at the cluster center. E now contains only intercluster edges, and the task is to decide

Algorithm 4 ADDSPANNEREDGES(V, V_i, E, E_S, K, n)

1: **for** $v \in V - V_i$ **do**
2: **if** v has edges incident on V_i **then**
3: $e_v \leftarrow \text{SHORTEST}\big(E(v, n_v)\big)$
4: $E_S \leftarrow E_S \cup \{e_v\}$
5: $E \leftarrow E - E(v, n_v)$
6: **for** $c' \in K_{i-1}$ **do**
7: $e_c \leftarrow \text{SHORTEST}E(v, c')$
8: **if** WEIGHT(e_c) <WEIGHT(e_v) **then**
9: $E_S \leftarrow E_S \cup \{e_c\}$
10: $E \leftarrow E - E(v, c')$
11: **end if**
12: **end for**
13: $n_v \leftarrow n_v \cup \{v\}$
14: **else**
15: **for** $c \in K_{i-1}$ **do**
16: $E_S \leftarrow E_S \cup \text{SHORTEST}\big(E(v, c)\big)$
17: $E \leftarrow E - E(v, c)$
18: **end for**
19: **end if**
20: **end for**

Algorithm 5 CLUSTERJOINING (K, E, a, E_S)

1: **if** a is odd **then**
2: $i \leftarrow \lfloor \frac{a}{2} \rfloor$
3: **else**
4: $i \leftarrow \lfloor \frac{a}{2} \rfloor - 1$
5: **end if**
6: **for** $c \in K_{\lfloor \frac{a}{2} \rfloor}$ **do**
7: **for** $c' \in K_i$ **do**
8: $E_S \leftarrow E_S \cup \text{SHORTEST}\big(E(c, c')\big)$
9: **end for**
10: **end for**

which of these must be added to the spanner. In Algorithm 5, only the shortest edge to each neighboring cluster is added. To maintain the spanner property, clusters from the previous iteration must also be considered when a is even (line 4).

In this manner, a sparse roadmap spanner can be constructed from a dense roadmap. Nevertheless, constructing an entire dense roadmap before removing edges can be wasteful. Every edge in the dense roadmap has been checked for collisions, but many of these will be discarded by the spanner algorithm. In the next section, an incremental spanner algorithm is described, which accepts or rejects edges as they are considered for addition to the roadmap and before collision checking is performed. It sacrifices asymptotic time complexity for a reduction in the number of collision checks, which is frequently the most expensive operation in motion planning.

B. Incremental Roadmap Spanner

The `IRS` (see Algorithm 6) takes the idea of the `SRS` one step further. Here, roadmap and spanner construction are interleaved. When the roadmap algorithm adds an edge, the spanner algorithm can reject it before collision detection is performed. Before discussing the implementation of the algorithm, some

Algorithm 6 INCREMENTALROADMAPSPANNER(t)

1: $V \leftarrow \emptyset, E \leftarrow \emptyset$
2: **while** !STOPPINGCRITERION() **do**
3: $v \leftarrow$ SAMPLEFREE()
4: $V \leftarrow V \cup \{v\}$
5: $k \leftarrow \lceil e \cdot (1 + \frac{1}{d}) \log(\|V\|) \rceil$
6: $U \leftarrow \text{NEAR}(V, v, k)$
7: sort U by non-decreasing distance from v
8: **for all** $u \in U$ **do**
9: **if** SHORTESTPATH(V, E, v, u) $>$ $t \cdot$WEIGHT(v, u) **then**
10: **if** COLLISIONFREE(v, u) **then**
11: $E \leftarrow E \cup \{(v, u)\}$
12: **end if**
13: **end if**
14: **end for**
15: **end while**
16: **return** (V, E)

subroutines must be defined for the particular C-space being worked on:

SAMPLEFREE uniformly samples a random configuration in C_{free}.

NEAR (V, v, k) returns the k configurations in set V that are nearest to configuration v with respect to a metric function. This can be implemented as a linear search through the members of V or as something more involved, such as a nearest neighbors search in a kd-tree [39].

COLLISIONFREE (v, u) detects if there is a path between configurations v and u in C_{free}. Generally, a local planner plots a curve in C from v to u. Points along this curve are tested for membership in C_{free}, and if any fail, the procedure returns `false`.

STOPPINGCRITERION determines when to stop looking for a solution. Some potential stopping criteria are

1) a solution of sufficient quality has been found;
2) allotted time or memory have been exhausted;
3) enough of the C-space has been covered;

or some combination of these or other criteria.

WEIGHT (v, u) returns the positive edge weight of (v, u). In the context of motion planning, the weight of an edge is frequently the cost of moving the robot from configuration v to configuration u along the curve provided by a local planner.

SHORTESTPATH (V, E, v, u) returns the cost of the shortest path between v and u. Note that the actual shortest path cost is not required, just whether it is larger than t times the weight of the edge (v, u). Instead of naively applying a full graph search, a length limited variation of A* search can be employed. In this variation, the search is aborted if a node with a cost value $f > t \times w(v, u)$ is expanded. When this happens, it is known that there is no path from v to u with an acceptable cost. Additionally, edges that connect two disconnected components can be added without doing any kind of search because the shortest path cost in this case is infinite.

First, the roadmap $G = (V, E)$ is initialized to empty (line 1). Then, it iterates until STOPPINGCRITERION returns `true` (line 2).

For each iteration, SAMPLEFREE is called and returns $v \in C_{\text{free}}$ (line 3). The sampled vertex is added to the roadmap (line 4). The number of nearest neighbors is calculated (line 5). The set U of the k-nearest neighbors of v is found by calling NEAR(V, v, k) (line 6). If NEAR does not already return U ordered by distance from v, then it must be sorted on line 7. For each potential edge connecting v to a neighbor in U, the inclusion criteria must be met before it is added to the roadmap. First, if a path exists between v and u with cost less than t times the weight of (v, u), then that edge can be rejected because it contributed little to path quality (line 9). Second, the edge must be checked for collision (line 10), as it is typically the case in the PRM framework. If the local planner does not succeed in finding a curve in C_{free}, the edge is rejected. If the edge passes both inclusion tests, it is added to the roadmap (line 11). Then, the next iteration is started.

A notable departure that IRS makes from Algorithm 1 is that the edges are not ordered globally. This ordering is not required to preserve solution quality, however, as will be shown later. A *local* ordering of potential edges is performed on line 7. This does not affect the theoretical bounds on solution quality but can be seen as a heuristic that improves the sparsity of the final roadmap.

Since the spanner property is tested before the collision check, many expensive collision checks can be avoided. This property greatly improves running time for practically sized roadmaps.

IV. ANALYSIS

The properties of SRS and IRS are formally described in this section.

A. Asymptotic Near-Optimality

Theorem 1: SRS is asymptotically near-optimal.

Proof: Asymptotic near-optimality (see Definition 3) differs from asymptotic optimality (see Definition 2) only in that the solution costs must converge to within t times the optimal cost instead of the actual optimal cost.

It is known that solutions on roadmaps produced by k-PRM* converge to the optimal costs [12]. A t-spanner of such a roadmap is guaranteed to have solutions within t times the cost of those in the original roadmap. SRS produces t-spanners of roadmaps constructed by k-PRM*. Therefore, solutions on roadmaps produced by SRS converge to within t times the cost of optimal, as the construction of the original roadmap by k-PRM* goes to infinity. ■

Theorem 2: IRS is asymptotically near-optimal.

Proof: This proof also relies on the asymptotic optimality of k-PRM* on which IRS is based. The proposed technique has two differences from k-PRM*.

First, on line 7, the potential neighbors of a newly added vertex are ordered by nondecreasing distance from the new vertex. This would have no effect on the asymptotic optimality of k-PRM* because edges are rejected based solely on collisions with obstacles. The order that edges are tested for collisions has no effect on those tests. The second difference, which is shown on line 9, adds an additional acceptance criterion. This has the effect of making the edges in a roadmap output by IRS a subset of those output by k-PRM*.

Consider a pair of roadmaps $G = (V, E)$ returned by k-PRM* and $G_S = (V, E_S \subset E)$ returned by IRS. For each edge (v, u) in E/E_S, there was a path from v to u with a cost less than t times the weight of (v, u). This invariant is enforced by line 9. The shortest path σ in G between any two points $a, b \in V$ has cost c^*. This path may contain edges that are in E but not in E_S. For each of these edges (v, u), there exists an alternate path in G_S, with cost $c_{(v,u)} \leq t \cdot w(v, u)$. Therefore, there is a path σ_S between a and b in G_S with cost $\sum_{(v,u)}^{\sigma_S} c_{(v,u)} \leq t \cdot c^*$. In other words, since each detour is no longer than t times the cost of the portion of the optimal path it replaces, the sum cost of all of the detours will not exceed t times the total cost of the optimal path. This way, as construction time goes to infinity, the roadmap returned by IRS becomes near-optimal with probability 1. ■

B. Time Complexity

The time complexity analysis of SRS and IRS can be understood in relation to the complexity of k-PRM*. For each of the n iterations of k-PRM*, a nearest neighbors search and collision checking must be performed. An ϵ-approximate nearest neighbor search can be done in $\log n$ time producing $k(n) = \log n$ neighbors. The edge connecting the current iteration's sample to each of these neighbors must be checked for collision at a cost of $\log^d p$ time, where p is the number of obstacles. Therefore, the total running time of k-PRM* is $O\big(n \cdot (\log n + \log n \cdot \log^d p)\big)$, which simplifies to $O(n \cdot \log n \cdot \log^d p)$.

1) Time complexity breakdown for each of n iterations of k-PRM* (ignoring constant time operations):
   a) NEAR (ϵ-approximate): $\log n$
   b) For each $\log n$ neighbors:
      i) COLLISIONFREE: $\log^d p$.

The time complexity of SRS is adding to the complexity of k-PRM* the complexity of the graph-spanner algorithm ($O(kn)$). The linear complexity of the graph-spanner algorithm is dominated by that of k-PRM*. Thus, overall the complexity of SRS is $O(n \cdot \log n \cdot \log^d p)$.

1) Time complexity breakdown for each of n iterations of IRS (ignoring constant time operations):
   a) NEAR (ϵ-approximate): $\log n$
   b) Neighbor ordering: $\log n \cdot \log \log n$
   c) For each $\log n$ neighbors:
      i) COLLISIONFREE: $\log^d p$
      ii) SHORTESTPATH $> t \cdot$ WEIGHT: $t^d \log n \cdot \log(t^d \log n)$ (average case).

For IRS, two additional steps are performed. At every iteration, $k = \log n$ neighbors must be sorted by their distance to the sampled vertex. Many implementations of NEAR return the list of neighbors in order of distance, in which case, the cost of this operation can be zero. If this is not the case, however, the cost of sorting these k neighbors is $O(k \log k)$, and since $k = \log n$, the final result is $O\big(\log n \cdot \log(\log n)\big)$.

A shortest path search must also be performed for each potential edge. If done across the entire roadmap, this has a cost of $O(m \log n) = O(n \log^2 n)$, where m is the number of edges

and $m = n \log n$, since each node is connected to at most $\log n$ neighbors. The shortest path algorithm will expand only the vertices with path cost from v that is lower than $t \cdot w(v, u)$. This is due to the fact that the algorithm requires only knowledge about the existence of a path between v and u shorter than this bound. When $t = 1$, the number of nodes expanded by such a search is, at most, $k(n) \in O(\log n)$. Assuming a uniform sampling distribution, the expected number of nodes that may be expanded when $t > 1$ is proportional to $t^d \log n$ (based on the volume of a d-dimensional hyper-sphere). This brings the *expected* time complexity of IRS to

$$O(n \cdot (\log n + \log n \cdot \log \log n + \log n \cdot (\log^d p + t^d \log n \cdot \log (t^d \log n))))$$

which simplifies to $O(n \cdot \log n \cdot t^d \log n \cdot \log (t^d \log n))$, or, for fixed t and d:

$$O(n \cdot \log^2 n \cdot \log \log n)$$

which is asymptotically slower than k-PRM*. However, as will be shown experimentally, the very large constants involved in collision checking, which is avoided by IRS, makes IRS a computationally efficient alternative, at least for roadmaps of practical size.

C. Size Complexity

The number of edges in a spanner produced by Algorithm 3 is $O(an^{1+\frac{1}{a}})$ which also bounds the number of edges produced by SRS.

IRS is inspired by Algorithm 1, which produces $O(n)$ edges (with constants related to t and d). This bound does not hold for IRS itself. Without a global ordering of edges, there is no guarantee that the minimum spanning tree is contained within the spanner. Because of this, the space complexity cannot be bounded lower than that provided by k-PRM*, which is $O(n \log n)$. The experimental results, however, suggest that the dominant term is linear as the number of edges added at each iteration appear to converge to a constant.

V. Evaluation

All experiments were executed using the Open Motion Planning Library (OMPL) [40] on 2-GHz processors with 4 GB of memory. Four representative environments were chosen from those distributed with OMPL.

Alpha puzzle: C_{free} is a subset of $SE(3)$ and is highly constrained in this classical motion planning benchmark [see Fig. 3(a)].

Bug trap: A rod-shaped robot must orient itself to fit through the narrow passage and escape a 3-D version of the classical bug trap [$SE(3)$; see Fig. 3(b)].

Cubicles: Two floors of loosely constraining walls and obstacles must be navigated [$SE(3)$; see Fig. 3(c)].

Maze: A complex environment in a lower dimensional ($SE(2)$) space [see Fig. 3(d)].

For each of these environments and stretch factor combination, ten roadmaps with 50 000 vertices each were generated.

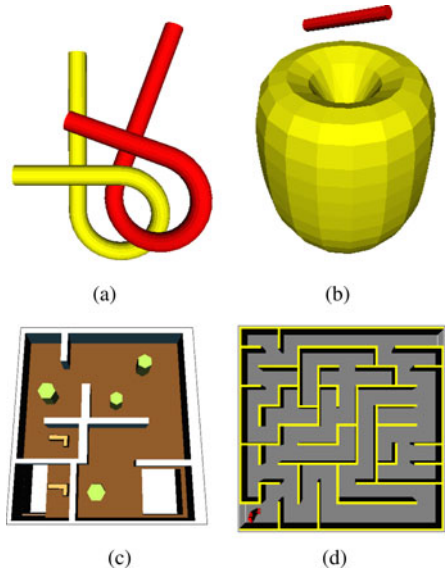


Fig. 3. Environments used in the experiments with example configurations for the robot. (a) Alpha puzzle. (b) Bug trap. (c) Cubicles. (d) Maze.

On each of these roadmaps, 100 random start and goal pairs were queried over the course of construction, and various qualities of the resulting solutions were measured. An additional $SE(3)$ environment (*apartment*) was considered in Fig. 4 as a representative case for high collision detection costs.

A. Construction Time

Although the expected asymptotic time complexity of IRS is worse than that of k-PRM*, the large constants involved in collision detection dominate the running time in many cases. Since IRS reduces the number of collision checks required at the cost of including a call to a graph search algorithm (due to the check on line 9), running time can be reduced for higher stretch factors. This is shown in Fig. 4, where a stretch factor of $t = 2$ allows IRS to construct a 50 000-node roadmap in under two-thirds of the time of k-PRM*. The diminishing returns shown for higher stretch factors reflect the larger area of the graph that must be searched for shortest paths. Construction times for the other experiments are shown in Fig. 5. Whether a time savings is achieved is dependent on features of the environment that affect collision checking such as the density of the obstacles and how expensive each collision check is to perform.

B. Space Requirements

The reduction in edges by the roadmap spanners is shown in Fig. 6. While each roadmap contains the same number of vertices (50 000), the space required for connectivity information is reduced by up to 85%. Although SRS has a lower asymptotic bound for the number of edges produced, IRS shows much better results, in practice, for the same stretch factor.

Fig. 7 shows the number of edges added to a roadmap as the number of nodes grows for both k-PRM* and IRS. In these environments, the linear term appears to dominate, even though the growth is bounded by $O(n \log n)$.

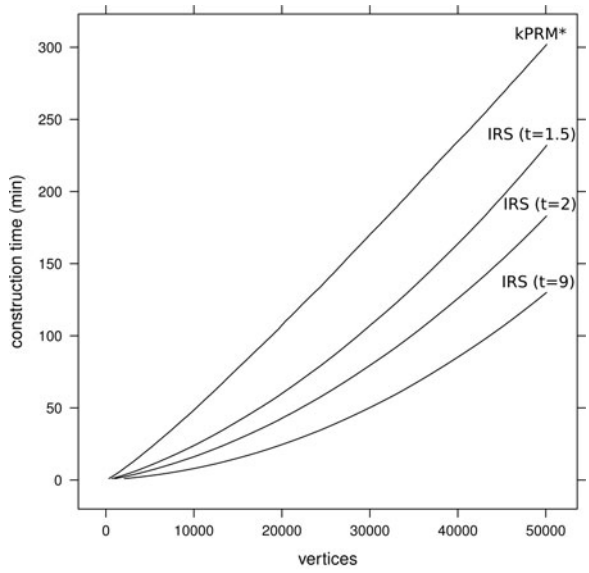


Fig. 4. Construction time and roadmap density for k-PRM* and IRS in the apartment environment (shown below). When the cost to perform collision detection is high, as in the complex apartment environment, practically sized roadmap spanners can be constructed faster because fewer edges must be checked for collision.

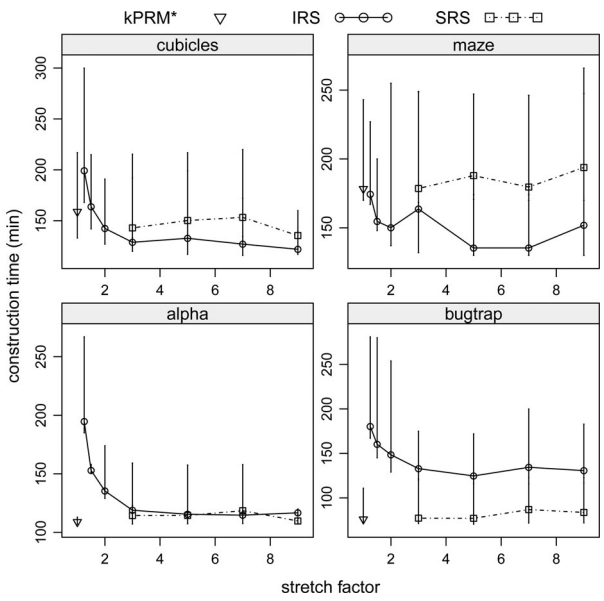


Fig. 5. Mean construction times of 50 000-node roadmaps for each environment. Error bars indicate the minimum and maximum times over ten runs.

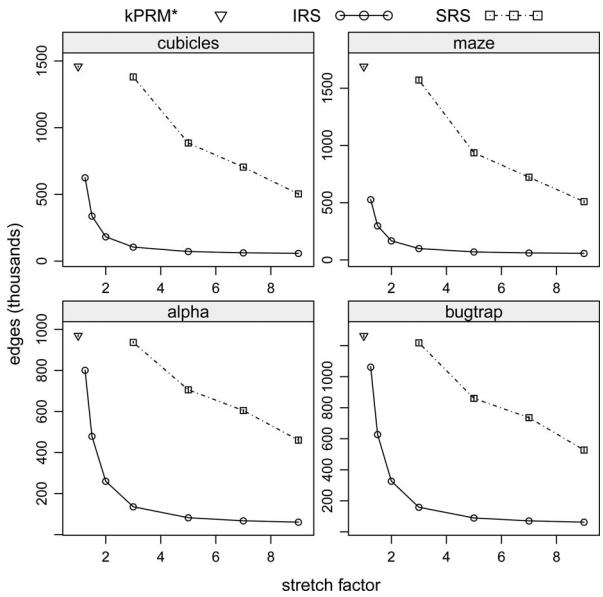


Fig. 6. Mean roadmap density of ten different 50 000-node roadmaps for each environment. The minimum and maximum values are within the size of the icons.

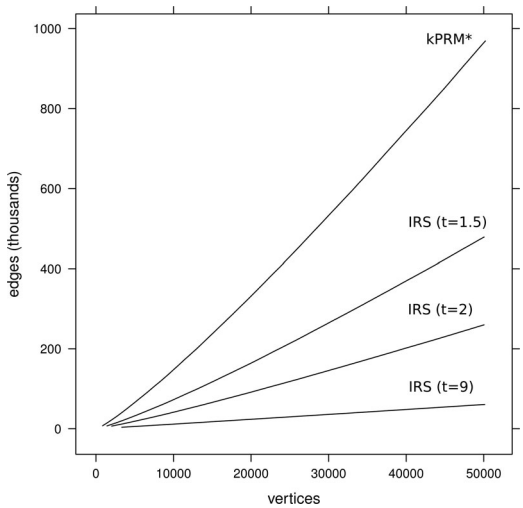


Fig. 7. Density growth for a roadmap constructed in the alpha environment. The reduction in the number of edges increases with stretch factor and grows approximately linearly with the number of vertices.

C. Solution Quality

In Fig. 8, path quality is measured by querying a roadmap with 100 random start and goal configurations. The lengths of the resulting paths increase as the number of edges in the spanner is reduced. It is important, however, to note that for these random starting and goal configurations, the average extra cost is significantly shorter than the worst case guaranteed by the stretch factor.

The worst degradation happens for short paths, where taking a detour of even a single vertex can increase the path length by a large factor. Path quality degradation in IRS is plotted in Fig. 9 as a function of its length in a roadmap generated by k-PRM*. All shortest paths to ten random vertices are averaged

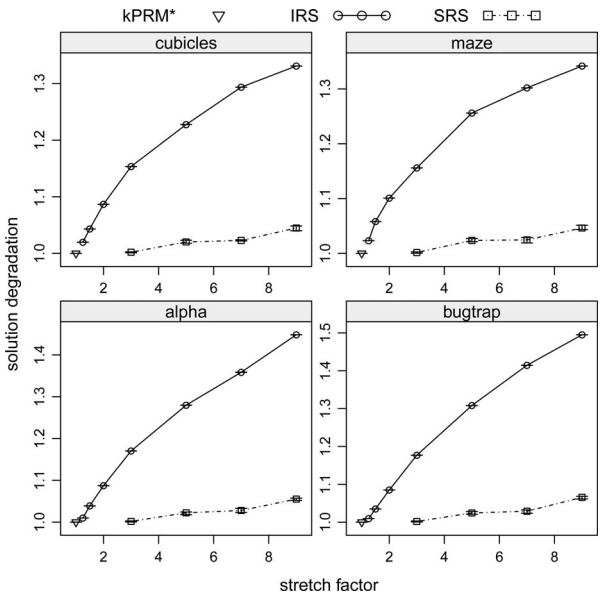


Fig. 8. Mean solution length for 100 random query pairs on ten different 50 000-node roadmaps compared with the k-PRM* outcome. Path length increases when the stretch factor increases. The increase is much less than the theoretical guarantee.

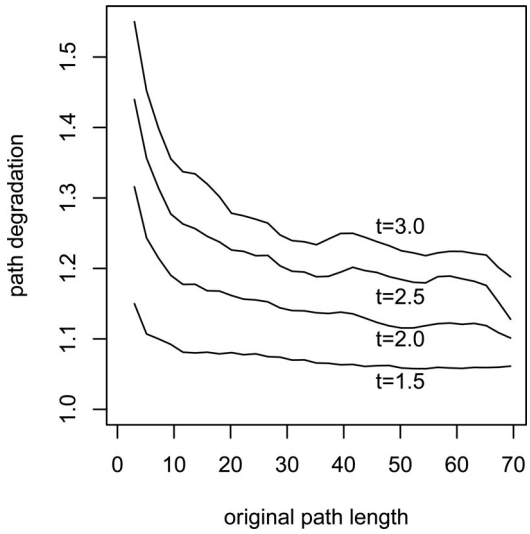


Fig. 9. Mean path length degradation for different stretch factors as a function of the path length in the original roadmap in the bug-trap environment with 5000 vertices. All shortest paths starting from ten random vertices in five different roadmaps were measured.

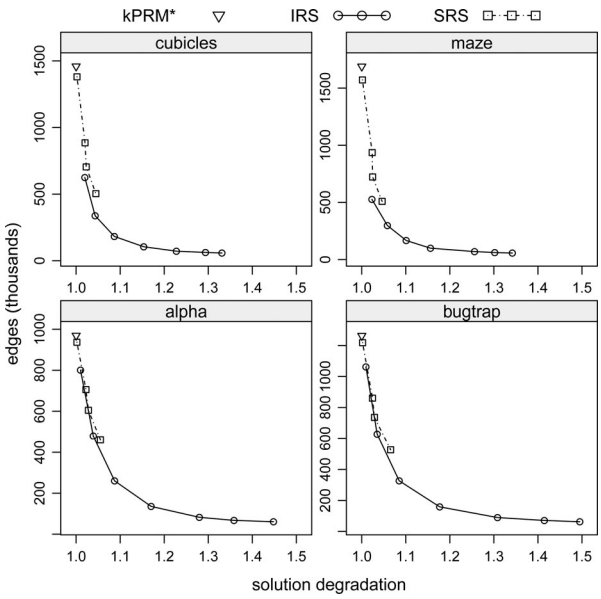


Fig. 10. Tradeoff between solution length and roadmap density averaged over 100 random query pairs on ten different 50 000-node roadmaps for each environment. Each data point for IRS and SRS represents a different stretch factor from 1.25 to 9. Except for the alpha environment, IRS dominates SRS in these two performance measures.

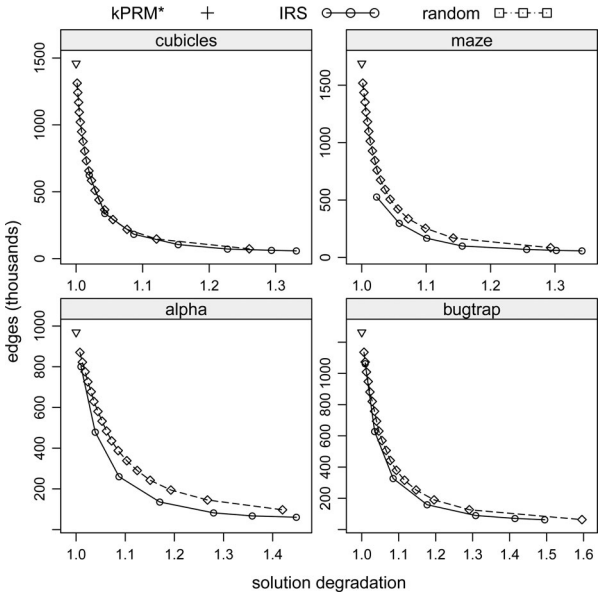


Fig. 11. Tradeoff between solution length and roadmap density averaged over 100 random query pairs on ten different 50 000-node roadmaps for each environment. Each data point for IRS represents a different stretch factor from 1.25 to 9. Each data point for the "random" method represents the probability an edge being removed from 5% to 90%. IRS dominates this random method in these two performance measures.

over five different roadmaps. In Figs. 10 and 11, the tradeoff between roadmap density and path quality is compared. It can be observed from these figures that IRS dominates both SRS and a naïve approach where random edges are removed. Table III provides the mean roadmap construction time in minutes for the algorithms in the different environments.

Overall, the results show that the SRS is not able to provide a large reduction in edges for low stretch values. For example, in one environment, it was only able to reduce the edge count by 15.3% for a stretch factor of 3. For higher stretch factors, e.g., up to 17, most of the edges (77% to 85%) can be removed by SRS, while increasing mean path length by a small amount (10% to

TABLE III
MEAN CONSTRUCTION TIME (MINUTES) FOR ALGORITHMS IN DIFFERENT ENVIRONMENTS

algorithm	alpha	bugtrap	cubicles	maze
k-PRM*	109	76	159	178
SRS	114	81	146	185
IRS	136	145	145	152
random	113	80	142	171

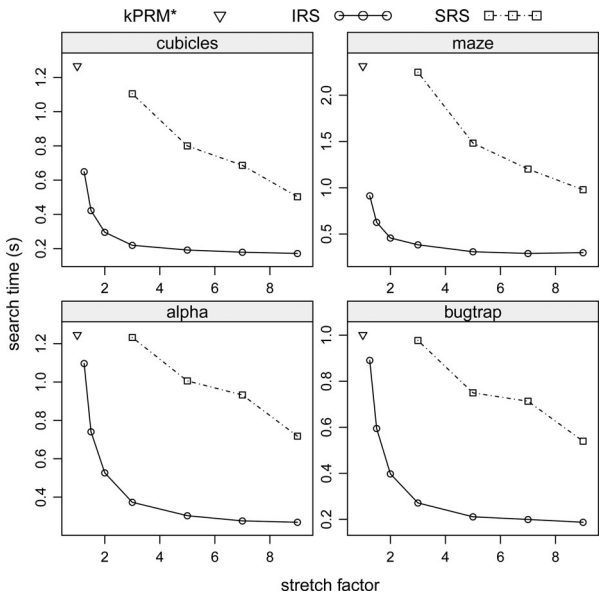


Fig. 12. Mean search time for 100 random query pairs on ten different 50 000-node roadmaps for each environment. Higher stretch factors produce roadmaps with fewer edges that can be searched more quickly.

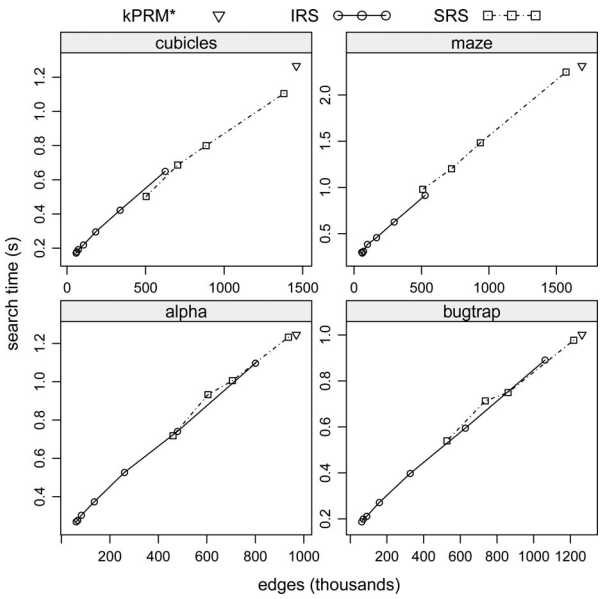


Fig. 13. In these environments (and in the environment shown in Fig. 4), roadmap search time has an approximately linear relationship to the density of the roadmap.

25%). The IRS described in Algorithm 1 was able to achieve a 70.5% edge reduction for relatively small stretch factors.

D. Roadmap Search Time

The time it took to perform an A* search on the resulting roadmap was also measured. The reported value does not include the time it takes to connect the start and goal configurations. As shown in Fig. 12, removing edges from the roadmaps reduces the query resolution time by up to 70%.

While reduced online query resolution time is presented here as a separate benefit, it is dependent on the size of the generated roadmap. Larger roadmaps generally result in longer search times. Experiments suggest that search time has an approximately linear relationship with the number of edges in roadmaps of these environments. This is illustrated in Fig. 13 and explains the similarity between Figs. 6 and 12.

E. Effects of Smoothing

For each query, a simple approach for path smoothing was tested. Nonconsecutive vertices on the path that were near each other were tested for potential collision-free connectivity. If they could be connected, then the path is shortened by removing the intervening vertices. This is a greedy and local method for smoothing, but it can be executed very quickly and produces impressive results. In Fig. 14, the smoothing factor is reported as the ratio of the smoothed cost to the unsmoothed cost. The time taken to smooth the solutions was between 0.01 and 0.7 s. A smoothing factor of less than one indicates that there was some improvement due to smoothing. Note that k-PRM* has yet to converge to the optimal solution; therefore, applying smoothing still improves the solution path. The solutions from roadmap spanners constructed with higher stretch factors showed more improvement after smoothing, reducing their cost relative to the smoothed k-PRM* solutions.

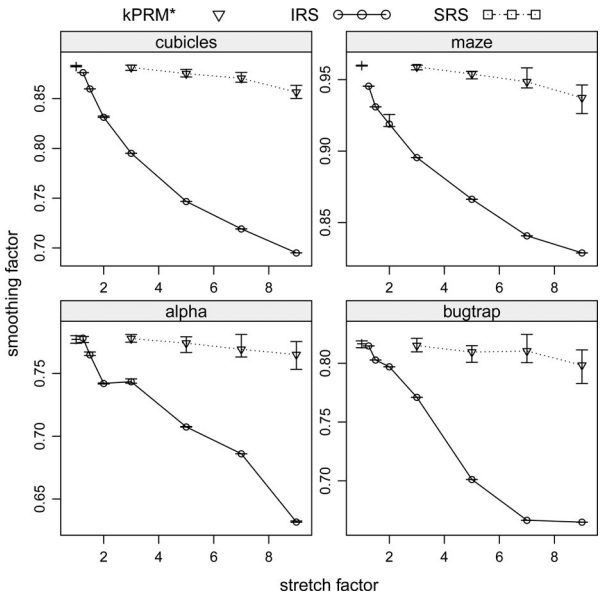


Fig. 14. Ratio of the cost of the smoothed solution to that of the unsmoothed one for 100 random query pairs on ten different 50 000-node roadmaps for each environment.

VI. Discussion

A. Selecting a Stretch Factor

The stretch factor parameter t should be selected based on the requirements of the system at hand. When path quality is of the utmost importance, a low value close to 1 should be used. For tasks that can afford lower path quality but need to reduce other

metrics such as preprocessing time, roadmap size, or online search time, a higher value should be used.

If preprocessing time is not an issue, then the parameter space can be searched until the highest value for t is found that produces a roadmap that fits into the space or search time allowed. Some tasks will have a natural method for optimizing the stretch factor. Define the total cost of completing a motion planning query as the time to search the roadmap (a) plus the time it takes to travel the path (b). Increasing t will decrease a, but increase b. Decreasing t will have the opposite affect. Then, the stretch factor t can be optimized to minimize the worst case for $a + b$.

If limits on detour length relative to a known optimum can be defined, then the stretch factor can be chosen directly. For example, if a path that can be followed by a robot is twice as long as another, the robot can be prevented from taking the undesirable path by setting t to less than 2.

B. Path Quality Guarantees Versus Actual Average Path Quality

The path quality guarantees in this study are based on the worst possible situation where every edge along an optimal path has been replaced by a detour that is t-times longer. While this is possible, it has a very low probability of happening for any query that must traverse a significant portion of the roadmap. For very short paths, that will utilize fewer roadmap edges, it is more likely. Nevertheless, the fact that some edges have, necessarily, not been removed pushes the average path degradation lower than the stretch factor limit, especially for queries with longer optimal paths. A practical measure of roadmap path quality might be the average degradation over all possible paths. In general, this is difficult to predict before actually constructing a roadmap but may be possible to calculate for environments with simple cost functions and no obstacles.

C. Applicability to Problems With Arbitrary Cost Functions

In order to guarantee near-optimal solutions, the cost function used in the methods presented here must be Lipschitz continuous. Such a function is relatively easy to construct for kinematic systems. If no guarantees are required, then an arbitrary cost function may be employed. The practical benefits of faster construction and search time, smaller roadmap than `k-PRM`*, and better path quality than `k-PRM` should still be present with most cost functions. If the cost function does not create a metric space, then a graph search algorithm other than A* should be used. Any work on asymptotically optimal planners that operate with arbitrary cost functions moves in an orthogonal direction and should be possible to integrate with the current work on roadmap spanners.

VII. Conclusion

This study has shown that it is practical to compute sparse roadmaps in C-spaces that guarantee asymptotic near-optimality. These roadmaps have considerably fewer edges than roadmaps with asymptotically optimal paths, while resulting in small degradation in path quality relative to the asymptotically optimal ones. The stretch factor parameter provides the ability to tune this tradeoff. The experiments confirm that roadmaps with low stretch factors have high path quality but are denser.

The current approach removes only roadmap *edges*. It is also the case, however, that nodes of the roadmap are redundant for the computation of near-optimal paths. Consecutive work to this one is investigating how to remove nodes so that the quality of a path answering a query in the continuous space is guaranteed not to get worse than a stretch factor [41]–[43].

It is important to study the type of near-optimality guarantees that can be provided in finite time, as in practice, a stopping criterion is employed to stop sampling-based algorithms. Furthermore, identifying the expected path quality degradation of a roadmap spanner will be a better indication of practical performance. Another direction to consider is to evaluate these algorithms in higher dimensional challenges, such as planning of articulated robots. The current experiments do not trend in a way that would suggest that problems would arise and the theoretical analysis covers these challenges.

Finally, it is important to study the relationship of roadmap spanners with methods that guarantee the preservation of the homotopic path classes in the C-space [10]. Intuitively, homotopic classes tend to be preserved by the spanner because the removal of an important homotopic class will have significant effects on the path quality.

Acknowledgment

The authors would like to thank the anonymous reviewers and the Associate Editor of the IEEE Transactions on Robotics for their constructive comments and feedback.

References

[1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3D workspaces," in *Proc. Workshop Algorithm. Found. Robot.*, 1998, pp. 155–168.

[3] L. J. Guibas, C. Holleman, and L. E. Kavraki, "A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Oct. 1999, vol. 1, pp. 254–259.

[4] T. Simeon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning," *Adv. Robot. J.*, vol. 41, no. 6, pp. 477–494, 2000.

[5] G. Sanchez and J.-C. Latombe, "A single-query, bi-directional probabilistic roadmap planner with lazy collision checking," in *Proc. Int. Symp. Robot. Res.*, 2001, pp. 403–418.

[6] D. Xie, M. Morales, R. Pearce, S. Thomas, J.-L. Lien, and N. M. Amato, "Incremental map generation (IMG)," presented at the Workshop Algorithm. Found. Robot., New York, Jul. 2006.

[7] R. Geraerts and M. H. Overmars, "Reachability-based analysis for probabilistic roadmap planners," *J. Robot. Autonom. Syst.*, vol. 55, pp. 824–836, 2007.

[8] R. Wein, J. van den Berg, and D. Halperin, "Planning high-quality paths and corridors amidst obstacles," *Int. J. Robot. Res.*, vol. 27, no. 11–12, pp. 1213–1231, Nov. 2008.

[9] B. Raveh, A. Enosh, and D. Halperin, "A little more, a lot better: Improving path quality by a path-merging algorithm," *IEEE Trans. Robot.*, vol. 27, no. 2, pp. 365–370, Apr. 2011.

[10] L. Jaillet and T. Siméon, "Path deformation roadmaps: Compact graphs with useful cycles for motion planning," *Int. J. Robot. Res.*, vol. 27, nos. 11/12, pp. 1175–1188, 2008.
[11] E. Schmitzberger, J. L. Bouchet, M. Dufaut, D. Wolf, and R. Husson, "Capture of homotopy classes with probabilistic roadmap," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2002, vol. 3, Lausanne, Switzerland, pp. 2317–2322.
[12] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.
[13] D. Nieuwenhuisen and M. H. Overmars, "Using cycles in probabilistic roadmap graphs," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr./May 2004, vol. 1, pp. 446–452.
[14] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Trans. Robot. Autom.*, vol. 21, no. 4, pp. 587–608, Aug. 2005.
[15] R. Geraerts and M. H. Overmars, "Creating small graphs for solving motion planning problems," in *Proc. IEEE Int. Conf. Meth. Models Autom. Robot.*, 2005, pp. 531–536.
[16] M. A. Morales, R. Pearce, and N. M. Amato, "Metrics for analyzing the evolution of C-space models," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2006, pp. 1268–1273.
[17] J. Kim, R. A. Pearce, and N. M. Amato, "Extracting optimal paths from roadmaps for motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, Taipei, Taiwan, Sep. 14–19, 2003, pp. 2424–2429.
[18] R. Geraerts and M. H. Overmars, "Creating high-quality roadmaps for motion planning in virtual environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Beijing, China, Oct. 2006, pp. 4355–4361.
[19] R. Pearce, M. Morales, and N. Amato, "Structural improvement filtering strategy for PRM," presented at the Robot. Sci. Syst., Zurich, Switzerland, Jun. 2008.
[20] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot. Res.*, vol. 21, no. 3, pp. 233–255, Mar. 2002.
[21] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, May 2001.
[22] O. Nechushtan, B. Raveh, and D. Halperin, "Sampling-diagrams automata: A tool for analyzing path quality in tree planners," presented at the Workshop Algorithm. Found. Robot., Singapore, Dec. 2010.
[23] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Beijing, China, Oct. 2006, pp. 5369–5375.
[24] K. Bekris and L. Kavraki, "Informed and probabilistically complete search for motion planning under differential constraints," presented at the 1st Int. Symp. Search Tech. Artif. Intell. Robot., Chicago, IL, Jul. 2008.
[25] Y. Li and K. E. Bekris, "Learning approximate cost-to-go metrics to improve sampling-based motion planning," presented at the IEEE Int. Conf. Robot. Autom., Shanghai, China, May 9–13, 2011.
[26] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," presented at the IEEE Int. Conf. Robot. Autom., Shanghai, China, May 2011.
[27] J. D. Marble and K. E. Bekris, "Computing spanners of asympotically optimal probabilistic roadmaps," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, San Francisco, CA, Sep. 2011, pp. 4292–4298.
[28] J. D. Marble and K. E. Bekris, "Asymptotically near-optimal is good enough for motion planning," presented at the Symp. Robot. Res., Flagstaff, AZ, Aug. 2011.
[29] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Trans. Comput.*, vol. C-32, no. 2, pp. 108–120, Feb. 1983.
[30] D. Peleg and A. Schäffer, "Graph spanners," *J. Graph Theor.*, vol. 13, no. 1, pp. 99–116, 1989.
[31] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares, "On sparse spanners of weighted graphs," *Discr. Comput. Geometr.*, vol. 9, no. 1, pp. 81–100, 1993.
[32] S. Baswana and S. Sen, "A Simple and linear time randomized algorithm for computing sparse spanners in weighted graphs," *Random Struct. Algorithm.*, vol. 30, no. 4, pp. 532–563, Jul. 2007.
[33] J. M. Keil, "Approximating the complete Euclidean graph," in *Proc. 1st Scandinav. Workshop Algorithm Theor.*. London, U.K.: Springer-Verlag, 1988, pp. 208–213.
[34] J. Gao, L. J. Guibas, and A. Nguyen, "Deformable spanners and applications," *Comput. Geometr. Theor. Appl.*, vol. 35, nos. 1/2, pp. 2–19, Aug. 2006.
[35] E. Cohen, "Fast algorithms for constructing t-spanners and paths with stretch t," *SIAM J. Comput.*, vol. 28, no. 1, pp. 210–236, 1998.
[36] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, "Near-linear time construction of sparse neighborhood covers," *SIAM J. Comput.*, vol. 28, no. 1, pp. 263–277, 1998.
[37] L. Roditty, M. Thorup, and U. Zwick, "Deterministic constructions of approximate distance oracles and spanners," in *Proc. Int. Colloq. Autom. Lang. Programm.*, 2005, pp. 261–272.
[38] M. Thorup and U. Zwick, "Approximate distance oracles," *J. Assoc. Comput. Mach.*, vol. 52, no. 1, pp. 1–24, Jan. 2005.
[39] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. Assoc. Comput. Mach.*, vol. 18, no. 9, pp. 509–517, Sep. 1975.
[40] I. A. Şucan, M. Moll, and L. E. Kavraki. (2013). The open motion planning library. *IEEE Robot. Autom. Mag.* [Online]. Available: http://ompl.kavrakilab.org
[41] J. D. Marble and K. E. Bekris, "Towards small asymptotically near-optimal roadmaps," presented at the IEEE Int. Conf. Robot. Autom., St. Paul, MN, May 14–18, 2012.
[42] A. Dobson, A. Krontiris, and K. E. Bekris, "Sparse roadmap spanners," presented at the Workshop Algorithm. Found. Robot., Cambridge, MA, Jun. 13–15, 2012.
[43] A. Dobson and K. E. Bekris, "Sparse roadmaps spanners for asymptotically near-optimal motion planning," *Int. J. Robot. Res.*, 2013, to be published.

James D. Marble (M'12) received the Bachelor's and Master's degrees in computer science and engineering from the University of Nevada, Reno, where he worked with Dr. K. Bekris on motion planning.

Specifically, his work has focused on near-optimal sampling-based algorithms. Toward this goal, he has contributed in the development of the PRACSYS open-source software platform for motion planning, replanning, and multirobot coordination. Since the summer of 2012, he has been with the Sierra Nevada Corporation, Sparks, NV, as a Software Engineer.

Kostas E. Bekris (M'07) received the Bachelor's degree in computer science from the University of Crete, Crete, Greece, in 2001 and the Master's and Doctoral degrees in computer science from Rice University, Houston, TX, in 2004 and 2008, respectively, where he worked with Prof. L. Kavraki.

Until June 2012, he served as an Assistant Professor with the Department of Computer Science and Engineering, University of Nevada, Reno. Since July 2012, he has been with the Department of Computer Science, Rutgers University, as an Assistant Professor. He is heading the PRACSYS group (http://www.pracsylab.org), which is part of the Computational Biomedicine, Modeling, and Imaging Center. His research interests include optimal motion planning, planning for systems with challenging dynamics, as well as multirobot planning and applications in robotics, cyber-physical systems, simulations, and games. His research lab has been supported by the National Science Foundation, the Office of Naval Research, and the National Aeronautics and Space Administration.

Dr. Bekris co-chairs the IEEE Robotics and Automation Society Technical Committee on Algorithms for Planning and Control of Robot Motion.