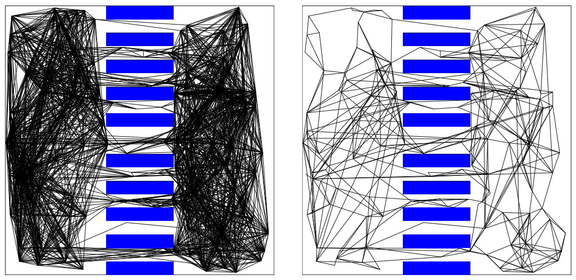
Computing Spanners of Asymptotically Optimal Probabilistic Roadmaps

James D. Marble and Kostas E. Bekris

***Abstract*— Asymptotically optimal motion planning algorithms guarantee solutions that approach optimal as more iterations are performed. Nevertheless, roadmaps with this property can grow too large and unwieldy for fast online query resolution. In graph theory there are many algorithms that produce subgraphs, known as spanners, which have guarantees about path quality. Applying such an algorithm to a dense, asymptotically optimal roadmap produces a sparse, asymptotically near optimal roadmap. Experiments performed on typical, geometric problems in SE(3) show that a large reduction in roadmap edges can be achieved with a small increase in path length. Online queries are answered much more quickly with similar results in terms of path quality. This also motivates future work that applies the technique incrementally so edges that won't increase path quality will never be added to the roadmap and won't be checked for collisions.**



(a) Roadmap with 2346 edges (b) Spanner with 470 edges

Fig. 1. A roadmap with 150 milestones that contains 2406 edges (a). The 17–spanner ($k = 9$) constructed from this roadmap (b), contains only 465 edges but preserves homotopy and adds only 7.3% to the mean path length over all shortest paths.

I. Introduction

Roadmap planners [1] utilize an off-line phase to build up knowledge about the configuration space ($\mathcal{C}$-space) before queries can be answered and solve many practical planning problems in complex $\mathcal{C}$-spaces. Path quality can be measured in terms of length, clearance, or smoothness [2]. Traditionally, many techniques focus on feasibility and may return paths of low quality; considerably different from the optimum ones. Smoothing can be used to improve some measures of path quality and algorithms exist that produce roadmaps with paths that are deformable to optimal ones [3], [4]. Hybridization graphs [5] combine multiple solutions into a higher quality one that uses the best portions of each input path. These techniques, however, can be expensive for the online resolution of a query, especially when multiple queries must be answered.

Alternatively, we can construct larger, denser roadmaps that better sample $\mathcal{C}$ by investing more preprocessing time. The k-nearest PRM* algorithm [6] minimizes the number of neighbors each new sample has to be connected to while guaranteeing increasing quality paths as more samples are added to the roadmap, a property known as asymptotic optimality. While asymptotically optimal roadmaps are desirable for their high path quality, their large size can be problematic. Large roadmaps impose significant costs during storage, transmission and online query resolution and may not be feasible for some applications.

In Section III of this paper, an algorithm is described that reduces the number of edges in a roadmap while providing guarantees about path quality. In Fig. 1, this is demonstrated by removing 80.7% of the edges. The maximum increase in path cost is guaranteed to be less than 1700%, but the mean increase in path cost (measured here as the average length of all shortest paths), in this instance, is only 7.3%. The spanner retains most of the beneficial properties of the original roadmap with one fifth the number of edges.

Such an operation can assist in any application where small, high quality roadmaps are preferable to large ones.

- Embedded computers used in robots are often resource constrained. Large, dense optimal roadmaps can be generated on a less constrained device offline. Then they can be trimmed while maintaining near optimal properties, transmitted quickly to a robot and more quickly queried than the full roadmap. This idea has applications in the emerging field of cloud robotics [7].
- Path planning for interactive games is given a small computational budget yet must produce believable results quickly. During development, however, an almost unlimited amount of time is available to produce an asymptotically near optimal graph of a size that can be quickly loaded and queried online during game play.
- Tree-based kinodynamic planners can use an approximate roadmap to estimate distances between states and the goal region that take into account $\mathcal{C}$-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 [8], [9].
- Edges of the roadmap can be dynamically invalidated given the position of obstacles in real-time [10], [11]. It must be possible to compute new paths on the fly from every location. Having a small roadmap is advantageous in this application.

The above tasks require a roadmap with the following properties:

- covers the $\mathcal{C}$-space (*coverage* property)

J. D. Marble and K. E. Bekris are with the Department of Computer Science and Engineering, University of Nevada, Reno, 1664 N. Virginia St., MS 171, Reno, NV, 89557 {jmarble,bekris}@cse.unr.edu

- is connected if the $\mathcal{C}$-space is connected (*connectivity* property)
- provides good approximations to optimal paths with a minimal need for smoothing (*high path quality* property)
- has small size and is sparse, so as to reduce storage and transmission requirements and the computational cost of online query resolution (*efficient query resolution* property).

Asymptotically near optimal roadmaps maintain coverage, connectivity, and high path quality from the original roadmap with the additional benefits of efficient query resolution attained by removing edges that contribute little to path quality.

A. Related Work

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 [12], [13], [14], [15], [16], [17], [18].

Tree-based algorithms [19], [20] focus on single query planning, but they do not provide the preprocessing properties of roadmaps. A tree is already a sparse graph and it becomes disconnected when removing edges. For this reason, the proposed technique can only be applied to planning algorithms that produce a dense roadmap. It has been shown that Bi-RRT produces arbitrarily bad paths with high probability and will miss high quality paths even if they are easy to find [21]. Anytime RRT [22] has been proposed as an approach to incrementally improve path quality.

Certain algorithms, such as the Visibility-based Roadmaps (Visib-PRM) [15], the Incremental Map Generation (IMG) algorithm [17] and the Reachability Roadmap Method (RRM) focus on returning a connected roadmap the covers the entire configuration space [23]. A reachability analysis of roadmap techniques suggested that connecting roadmaps is more difficult than covering $\mathcal{C}$-space [24].

A method is available to characterize the contribution of a sample to the exploration of the $\mathcal{C}$-space [25], but it does not address how the connectivity between samples contributes to the resulting path quality once the space has been explored.

Work on creating roadmaps that contain good quality paths has been motivated by the objective to efficiently resolve queries without the need for a post-processing optimization step of the returned path. One technique aims to compute all different homotopic solutions [4], while Path Deformation Roadmaps compute paths that are deformable to optimal ones [3]. Another approach inspired by Dijkstra's algorithms extracts optimal paths from roadmaps for specific queries [26] but may require very dense roadmaps. The idea of Useful Cycles has been used to create roadmaps with small number of edges that approximate optimal paths [27]. The Useful Cycles idea has been combined with the RRM to construct connected roadmaps that cover 2D and 3D $\mathcal{C}$-spaces and provide good quality paths [28].

The RRG, RRT*, and PRM* family of algorithms [6] provide asymptotic optimality for general configuration spaces. RRG and RRT* are based on RRT, a tree-based planner. The Anytime RRT* approach [29] extends RRT* with anytime planning in dynamic environments [10] and can incrementally improve path quality. PRM* is a modification of standard PRM. The proposed technique uses a variation of PRM*, k-nearest PRM* in the offline, roadmap building step.

TABLE I
SPANNER ALGORITHMS AND THEIR PROPERTIES

algorithm	stretch	edges	time
Althöfer et al. [35]	$2k-1$	$O(n^{1+\frac{1}{k}})$	$O(mn^{1+\frac{1}{k}})$
Cohen [32]	$2k+\epsilon$	$O(kn^{1+\frac{1}{k}})$	$O(mn^{\frac{1}{k}})$
Awerbuch et al. [33]	$64k$	$O(kn^{1+\frac{1}{k}})$	$O(mn^{\frac{1}{k}})$
Roditty et al. [36]	$2k-1$	$O(n^{1+\frac{1}{k}})$	$O(kn^{2+\frac{1}{k}})$
Thorup et al. [34]	$2k-1$	$O(kn^{1+\frac{1}{k}})$	$O(kmn^{\frac{1}{k}})$
Baswana et al. [37]	$2k-1$	$O(kn^{1+\frac{1}{k}})$	$O(km)$

The structural properties of a roadmap can be used to filter vertices that do not improve path quality [30]. This method is similar to that described in this paper, but provides no guarantees on the resulting path quality.

B. Contribution

The contribution of this work is a method for removing edges from a roadmap with path quality guarantees while retaining near optimal path quality utilizing ideas regarding the generation of spanner graphs [31]. Online query resolution time is shortened while simultaneously reducing the space required for storing and transmitting the roadmap.

The theoretical guarantees on path quality that this technique provides are tested empirically in a variety of motion planning problems in $SE(3)$. In all of these environments, most of the edges (77% to 85%) can be removed while increasing mean path length by only a small amount (10% to 25%). In some environments, this increase can be reduced (down to 2%) by utilizing path smoothing. This path degradation is most pronounced for paths that are very short. Longer, paths are less affected. Finally, a simple heuristic is described that can increase the performance of the spanner algorithm by a small amount.

II. GRAPH-THEORETIC FOUNDATIONS

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

Graph spanners, as formalized in [31], arose as the underlying data-structures of other, domain specific, algorithms. For example, algorithms computing approximate all-pairs shortest paths [32], [33], [34] are based, implicitly on the concept of spanners.

A number of spanner algorithms are available in the literature. Here, only algorithms without any restrictions on the type of graph they are applied to are considered. For example, many spanner algorithms operate on *complete*

Euclidean graphs [38]. These would preclude roadmaps that operate in $\mathcal{C}$-spaces with obstacles because obstacles remove some edges from the complete graph.

Small spanners (those with fewer edges) are desirable, but computing the smallest t-spanner for a given graph is NP-hard [31]. Even *approximating* such a spanner with one of size $O(2^{(1-\mu)\ln n})$ for $\mu > 0$ and $t > 2$ is NP-hard [39]. Because of this, existing algorithms seek to compute $(2k-1)$-spanners of size $O(n^{1+1\frac{1}{k}})$ (where k is a parameter related to the stretch).

Of the algorithms that provide spanners of this size and can be applied to general, weighted graphs, the performance characteristics vary (Table I). Of these, only one provides optimal stretch $(2k-1)$ along with reasonable size ($O(kn^{1+\frac{1}{k}})$) and linear time complexity ($O(km)$), where n is the number of vertices and m is the number of edges. The randomized $(2k-1)$-spanner algorithm [37] combines these properties with ease of implementation to make it a good match for the proposed technique.

III. Approach

The approach can be broken down into two parts. First, a dense, asymptotically optimal roadmap is constructed using k-nearest PRM*. The result is passed through the randomized $(2k-1)$-spanner algorithm to make the roadmap sparse by removing edges that don't contribute much to path quality.

A spanner algorithm could be applied to a roadmap produced by any variant of PRM. However, without the path quality guarantees provided by an asymptotically optimal planner, there would be no guarantees on the path quality of the sparse roadmap.

A. *k-nearest* PRM*

A robot can be abstracted as a point in a d-dimensional $\mathcal{C}$-space where collision with an obstacle is determined by testing if a particular configuration is part of $\mathcal{C}_{free}$. This is illustrated in Fig. 2, where the grey object represents an obstacle projected into $\mathcal{C}$. If a local planner is available that can connect two points in the $\mathcal{C}$-space, if possible, then PRM can be applied to the problem.

The standard PRM attempts to connect sampled configurations to either a fixed number, k, of nearest neighbors or all configurations within a fixed radius ball centered at the new configuration. Although construction can be stopped after a fixed amount of time or when the increase in path quality slows, it has been shown that a PRM using a fixed k does not converge to the optimal path [6].

The k-nearest PRM* algorithm (Algorithm 1) rectifies this by making k a logarithmic function of the size of the roadmap (line 1). It has been proven that roadmaps constructed with this variation will almost surely converge to optimum solutions [6].

B. *Randomized* $(2k-1)$ *Spanner Algorithm*

The inputs to the randomized $(2k-1)$-spanner (Algorithm 2) are the sets of vertices and weighted edges of the roadmap, V and E; and the spanner parameter, k. The output is E_S, the edges of the spanner. There are two parts to the algorithm. First, clusters are formed. Second, the clusters are joined to each other.

Algorithm 1 k-nearest PRM*

```
1: $(V, E) \leftarrow (\emptyset, \emptyset)$
2: while !StoppingCriteria() do
3:     $v \leftarrow$ SampleFree()
4:     $V \leftarrow V \cup \{v\}$
5:     k $\leftarrow \lceil e(1+\frac{1}{d})\log(\|V\|)\rceil$
6:     $U \leftarrow$ Near($V, E, v$, k)
7:     for all $u \in U$ do
8:         if CollisionFree($v, u$) then
9:             $E \leftarrow E \cup \{(v, u)\}$
```

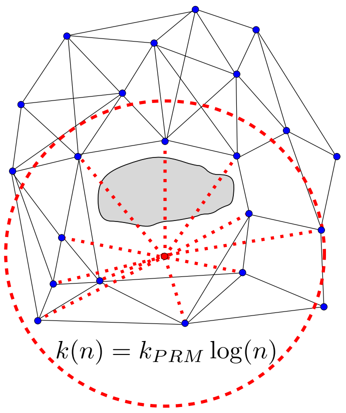


Fig. 2. The k-nearest PRM algorithm samples a random configuration (red) then attempts to connect it to the k-nearest vertices, where k is some constant. The k-nearest PRM* algorithm is a variation where k is a function of the number of vertices already added. Specifically, $k(n) = k_{PRM}\log n$, where $k_{PRM} > (1 + 1/d)$ and d is the dimensionality of the problem.

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 clustering, K_0. With each successive iteration, the following steps are taken:

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

This is repeated until $\lfloor\frac{k}{2}\rfloor$ iterations have been performed. In the second part of the algorithm, the clusters are joined by the shortest edge that connects them to their neighbors (Fig. 3(e)).

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

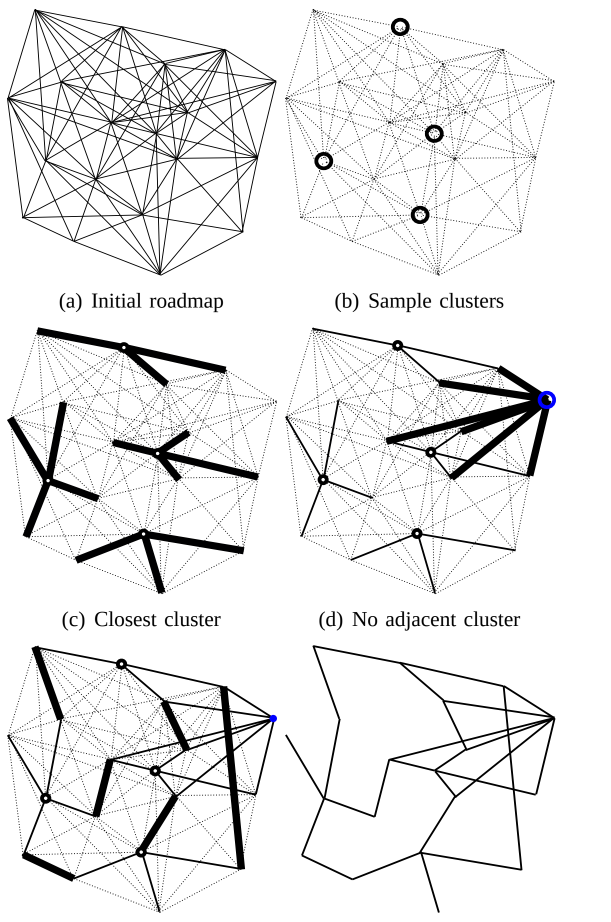

(a) Initial roadmap (b) Sample clusters
(c) Closest cluster (d) No adjacent cluster
(e) Cluster joining (f) Final spanner

Fig. 3. Illustration of the randomized $(2k-1)$-spanner algorithm for $k=2$.

Algorithm 2 RandomizedSpanner(V, E, k)

```
1:  $E_S \leftarrow \emptyset$
2:  $K_0 \leftarrow \{\{v\} | v \in V\}$
3:  for $i \in \{1, \ldots, \lfloor \frac{k}{2} \rfloor\}$ do
4:      $K_i \leftarrow \emptyset$
5:      for $c \in K_{i-1}$ do
6:          if UniformRandom$(0,1) < \|V\|^{-\frac{1}{k}}$ then
7:              $K_i \leftarrow K_i \cup \{c\}$
8:              $V_i \leftarrow V_i \cup c$
9:      for $v \in V - V_i$ do
10:         $n_v \leftarrow$ nearest cluster in $K_i$ to $v$
11:     AddSpannerEdges$(V, E, K_i, n)$
12:     for $e \in E$ do
13:         $(v, u) \leftarrow e$
14:         if $v$ and $u$ are in the same cluster of $K_i$ then
15:             $E \leftarrow E - \{e\}$
16: ClusterJoining$(K, E, k, E_S)$
17: return $E_S$
```

The clustering K_0 is initialized (line 2) so that each vertex becomes it's own cluster. Lines 2–2 randomly sample the last iteration's clusters with probability $n^{-\frac{1}{k}}$ (Fig. 3(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 2).

Sampled clusters are expanded in lines 2–2. Here, the closest, neighboring, sampled cluster, n_v, is found for each vertex that is not a member of a sampled cluster. Algorithm 3 adds edges to the spanner that connect the new vertices to their clusters.

Algorithm 3 AddSpannerEdges(V, E, K_i, n)

```
1:  for $v \in V - V_i$ do
2:      if $v$ has edges incident on $V_i$ then
3:          $e_v \leftarrow$ 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:              $e_c \leftarrow$ 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:         $n_v \leftarrow n_v \cup \{v\}$
12:     else
13:         for $c \in K_{i-1}$ do
14:             $E_S \leftarrow E_S \cup$ Shortest$\big(E(v, c)\big)$
15:             $E \leftarrow E - E(v, c)$
```

For those vertices that are adjacent to sampled clusters, lines 3–3 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 3–3).

If a cluster has no neighboring sampled clusters it is connected to each of the previous iteration's clusters by the shortest edge (Fig. 3(d)). Because this introduces extra edges, it is desirable to minimize the number of clusters that have no neighbors. Algorithm 5 (described later) introduces an alternative sampling method that attempts to do this.

At the conclusion of each iteration (lines 2–2, in Algorithm 2), intra-cluster edges are removed from E.

Algorithm 4 ClusterJoining (K, E, k, E_S)

```
1:  if $k$ is odd then
2:      $a \leftarrow \lfloor \frac{k}{2} \rfloor$
3:  else
4:      $a \leftarrow \lfloor \frac{k}{2} \rfloor - 1$
5:  for $c \in K_{\lfloor \frac{k}{2} \rfloor}$ do
6:      for $c' \in K_a$ do
7:          $E_S \leftarrow E_S \cup$ Shortest$\big(E(c, c')\big)$
```

After all $\lfloor \frac{k}{2} \rfloor$ iterations have completed, $K_{\lfloor \frac{k}{2} \rfloor}$ contains the final clusters, each of which is a tree rooted at the cluster center. E now contains only inter-cluster edges, and the task is to decide which of these must be added to the spanner. In Algorithm 4, 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 k is even (line 4).

C. Heuristic

The random sampling of clusters can result in uneven distribution of clusters at the local level where two or more clusters are very close together, wasting their individual potentials, and some vertices are not adjacent to any clusters forcing more of their edges to be added to the spanner.

Algorithm 5 Heuristic cluster sampling

```
while $\|K_i\| < \|V\|^{\frac{1}{k}}$ do
    $c \leftarrow \arg\max_{c \in K_{i-1}}$ UNSAMPLEDDEGREE$(c)$
    $K_i \leftarrow K_i \cup \{c\}$
    $V_i \leftarrow V_i \cup c$
```

To minimize the number of clusters not adjacent to sampled clusters, a heuristic sampling method was implemented (Algorithm 5). This can be substituted for lines 2–2 in Algorithm 2. Instead of randomly sampling clusters, they are chosen deterministically by selecting the cluster that maximizes the number of adjacent vertices that are not already next to a sampled cluster. This is done until the number of clusters reaches the expected amount given the randomized method ($n^{-\frac{1}{k}}$).

IV. Evaluation

All experiments were run using the Open Motion Planning Library (OMPL) [40] on a 3GHz Intel Core 2 Duo processor with 4GB of memory. Three representative environments were chosen from those distributed with OMPL.

easy: Two large areas in the free space are connected by a narrow passage (Fig. 4(a)). The movable object is small enough that it doesn't have to twist to get through the hole.

cubicles: Two floors of loosely constrained walls and obstacles (Fig. 4(b)).

bug trap: A rod shaped, movable object must orient itself to fit through the narrow passage and escape a three-dimensional version of the classical bug trap (Fig. 4(c)).

alpha puzzle: The entire free space is highly constrained in this classic motion planning benchmark (Fig. 4(d)).

Table II summarizes the results of 100 random queries on roadmaps with 20,000 vertices. The column labeled "original" shows the absolute value of each measure on the roadmap from k-nearest PRM*. Each successive column shows the results of running the randomized $(2k-1)$ spanner algorithm on the original roadmap with different values of k.

A. Space Requirements

While each spanner contains the same 20,000 vertices as the original roadmap, the space required for connectivity information is reduced by up to 85%. Environments with a more connected free space had a larger reduction in the number of edges because they had more edges that could be removed while still maintaining connectivity.

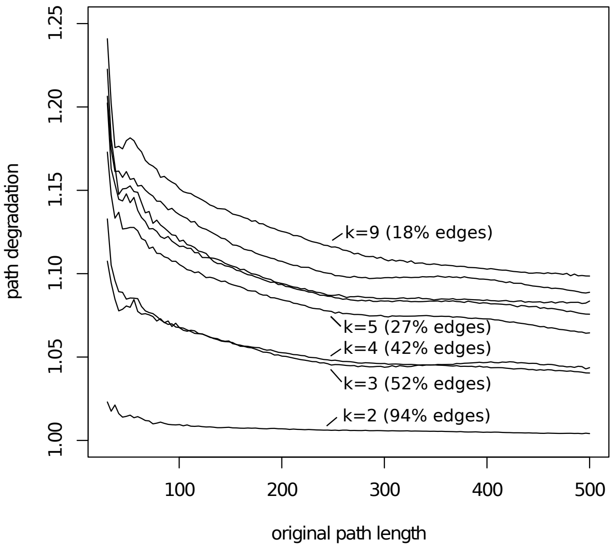


Fig. 5. Mean path length degradation as a function of the path length in the original roadmap in the "easy" environment with 20,000 vertices.

B. Path Quality

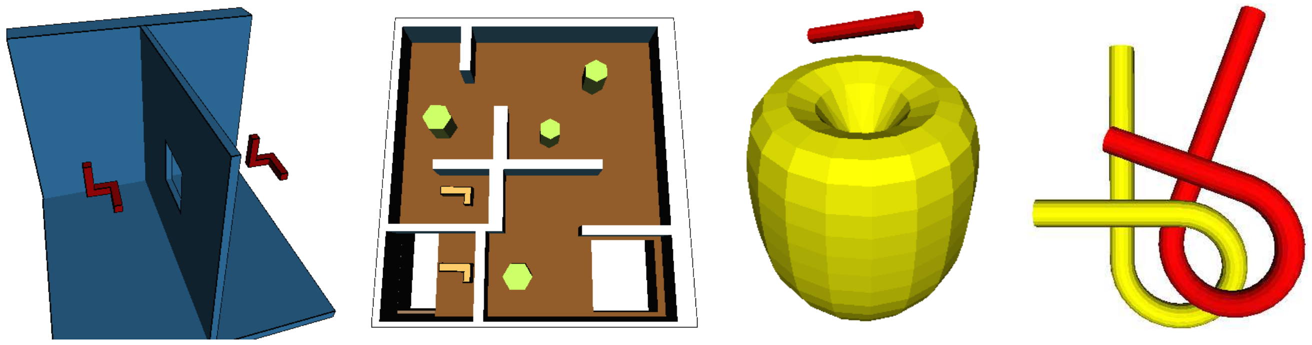
In Table II, path quality is measured by querying a roadmap with 100 random start and goal configurations. The length of the resulting paths increase as the number of edges in the spanner is reduced. This is expected because if an edge on a shortest path is removed then the new shortest path must take a necessarily longer detour. For these random starting and goal configurations, the extra cost is much shorter than the worst case guaranteed by the spanner algorithm.

It is interesting which paths are most affected by this increase in path length. Path quality degradation in the spanner roadmap is plotted in Fig. 5 as a function of the original path length and k. In this figure, many more paths are considered than for Table II. All shortest paths to 10 random vertices are averaged over 5 different roadmaps.

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. The path length for the 17–spanner ($k = 9$) in this experiment increases by a maximum of 200%, although this is for a very short path length and the mean at that length is less than 25%.

C. Query Resolution Time

Query resolution time includes the time it takes to connect the start and goal configurations to the roadmap and perform an A* search. The connection time is not effected by the number of edges in the roadmap, only the number of vertices. Since each spanner has the same number of vertices as the original roadmap, it can be seen from the results that the discrete search dominates the query resolution time for such a large roadmap. Reducing the number of edges in the roadmap lowers the query resolution time by up to 75%.



(a) easy (b) cubicles (c) bug trap (d) alpha puzzle

Fig. 4. Environments used in the experimental evaluation with example configurations for the movable object.

TABLE II

PERFORMANCE OF 100 RANDOM QUERY PAIRS AVERAGED OVER 5 RUNS ON 20,000 VERTEX ROADMAPS.

environment		original	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$
easy	edges (millions)	1.894	85.83%	38.13%	32.80%	21.05%	21.79%	16.58%	17.89%	14.81%
	solution length	281.39	100.25%	102.76%	102.98%	105.95%	105.35%	107.20%	105.86%	106.64%
	smoothed solution length	270.74	100.09%	100.38%	100.63%	101.47%	101.39%	101.57%	101.20%	100.43%
	query resolution time (s)	2.65	86.93%	43.76%	39.55%	29.68%	29.81%	25.51%	27.02%	24.21%
cubicles	edges (millions)	1.814	85.92%	37.98%	31.85%	21.50%	21.79%	17.03%	17.74%	14.97%
	solution length	320.12	121.78%	128.87%	127.61%	130.48%	129.32%	132.01%	131.35%	132.70%
	smoothed solution length	309.03	122.26%	127.20%	126.21%	126.73%	126.28%	127.23%	127.32%	126.77%
	query resolution time (s)	1.44	92.07%	45.07%	41.70%	32.00%	34.32%	28.99%	30.38%	27.62%
bug trap	edges (millions)	1.486	90.46%	46.91%	40.42%	26.69%	27.53%	20.98%	22.26%	18.46%
	solution length	26.44	101.64%	104.17%	104.87%	109.55%	107.19%	110.44%	107.95%	111.47%
	smoothed solution length	24.84	101.10%	101.31%	101.48%	103.96%	103.21%	104.26%	102.81%	103.25%
	query resolution time (s)	1.1	97.08%	56.50%	53.20%	41.22%	43.66%	38.15%	38.65%	35.35%
alpha	edges (millions)	1.113	91.90%	53.36%	46.80%	32.37%	32.55%	25.78%	27.27%	23.37%
	solution length	47.42	100.29%	103.33%	102.93%	105.43%	105.15%	106.85%	105.96%	107.79%
	smoothed solution length	44.52	100.03%	101.49%	101.18%	101.62%	102.41%	101.50%	102.20%	102.03%
	query resolution time (s)	1.43	102.17%	65.54%	62.26%	49.16%	50.36%	43.95%	45.27%	41.47%

D. *Effects of Smoothing*

For each query, there was an attempt at path smoothing. Nonconsecutive vertices on the path that were near each other were tested for connectivity. If they could be connected, then the path is shortened by removing intervening vertices. This is a greedy and local method for smoothing.

As illustrated in Fig. 1, a spanner can be expected to maintain the homotopic path classes inherited from the original roadmap. Paths in the same homotopic class as the optimal path can be smoothed to the same optimal path. The result of smoothing the shortest path in the spanners is not always the smoothed shortest path in the original roadmap in this case, however. In some environments, the mean degradation is less than 5% when $k = 9$.

It should be noted that the smoothing time increases up to 300% for paths on spanners with $k = 9$. This reflects the larger number of vertices in these paths. However, the time taken to smooth the solutions is three orders of magnitude shorter than the query resolution time in these experiments, although it may become consequential in other environments or for other smoothing techniques.

E. *Performance of Heuristic*

The heuristic used reduces the number of clusters that are not adjacent to sampled clusters. On the first iteration, this is beneficial because vertices not adjacent to sampled clusters must add all edges to the spanner, even ones that would not contribute to many short paths.

At later iterations, the heuristic is counterproductive, however, because clusters no longer correspond to single vertices and may have many inner vertices. Even if the cluster is adjacent to a sampled cluster, the inner vertices may not be and all of their edges will be added to the spanner.

This is reflected in Fig. 6, where low values of k show improvements for the heuristic over the randomized version while high values are worse.

V. DISCUSSION

This work shows that it is practical to compute sparse roadmaps in $\mathcal{C}$-spaces that guarantee asymptotically near optimal paths. The experimental results suggest that it is possible for these roadmaps to have considerably fewer edges than roadmaps with asymptotically optimal paths, while resulting in much smaller degradation in path quality relative to the almost optimal roadmaps.

The existing approach removes only edges from the original roadmap. Since, however, the roadmap is embedded in a continuous $\mathcal{C}$-space, it may be that nodes of the roadmap are redundant for the computation of near optimal paths. Future work will investigate how to remove nodes from roadmaps so

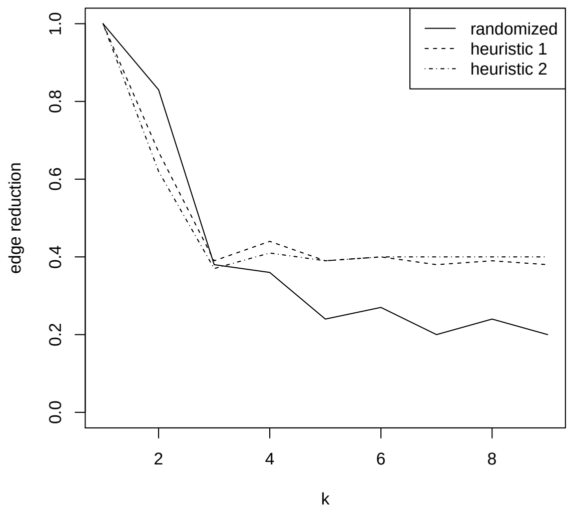


Fig. 6. Performance of the proposed heuristic averaged over 5 different runs on roadmaps with 2,000 vertices. Heuristic 1 is that described in Algorithm 5. Heuristic 2 is the same, but doubles the expected number of clusters.

that the quality of a path answering a query in the continuous space is guaranteed not to get worse than a specific threshold.

The approach described in this work requires the computation of a dense roadmap. An important direction is to develop methods for the computation of sparse roadmaps that directly provide asymptotically near optimal path guarantees. The cost of computing such a graph is considerably smaller than the cost of an algorithm such as k-nearest PRM* and also reduces the space requirements of the offline process [41].

Finally, it is important to study the relationship of the resulting spanner roadmaps with methods that guarantee the preservation of the homotopic path classes in the $\mathcal{C}$-space [3]. Intuitively, homotopic classes tend to be preserved by the spanner because the removal of an important homotopic class will have significant effects in the path quality.

References

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

[2] R. Wein, J. van den Berg, and D. Halperin, "Planning High-Quality Paths and Corridors amidst Obstacles," *IJRR*, vol. 27, no. 11-12, pp. 1213–1231, Nov. 2008.

[3] L. Jaillet and T. Simeon, "Path deformation roadmaps," in *WAFR*, New York City, NY, 2006.

[4] E. Schmitzberger, J. L. Bouchet, M. Dufaut, D. Wolf, and R. Husson, "Capture of homotopy classes with probabilistic roadmap," in *IROS*, Switzerland, 2002, pp. 2317–2322.

[5] B. Raveh, A. Enosh, and D. Halperin, "A Little More, a Lot Better: Improving Path Quality by a Path-Merging Algorithm," *Robotics, IEEE Transactions on*, vol. 27, no. 2, pp. 365–370, 2011.

[6] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *IJRR*, vol. 30, no. 7, pp. 846–894, June 2011.

[7] R. Arumugam, V. Enti, and L. Bingbing, "DAvinCi: A cloud computing framework for service robots," *ICRA*, pp. 3084–3089, 2010.

[8] K. Bekris and L. Kavraki, "Informed and Probabilistically Complete Search for Motion Planning under Differential Constraints," *TRA*, pp. 3–10, 2008.

[9] Y. Li and K. E. Bekris, "Learning approximate cost-to-go metrics to improve sampling-based motion planning," in *ICRA*, Shanghai, China, 9-13 May 2011.

[10] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *ICRA*, May 2006, pp. 2366–2371.

[11] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," in *ICRA*, vol. 5, no. April. IEEE, 2004, pp. 4399–4404.

[12] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: An Obstacle-based PRM for 3D Workspaces," in *WAFR*, 1998, pp. 155–168.

[13] 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 *IROS*, vol. 1, October 1999, pp. 254,259.

[14] D. Hsu, L. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin, "On finding narrow passages with probabilistic roadmap planners," in *WAFR*, Houston, TX, 1998.

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

[16] G. Sanchez and J.-C. Latombe, "A single-query, bi-directional probabilistic roadmap planner with lazy collision checking," in *ISRR*, 2001, pp. 403–418.

[17] D. Xie, M. Morales, R. Pearce, S. Thomas, J.-L. Lien, and N. M. Amato, "Incremental Map Generation (IMG)," in *WAFR*, New York City, NY, July 2006.

[18] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *TRA*, vol. 21, no. 4, pp. 587–608, 2005.

[19] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *IJRR*, vol. 20, no. 5, pp. 378–400, May 2001.

[20] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock, "Randomized Kinodynamic Motion Planning with Moving Obstacles," *IJRR*, vol. 21, no. 3, pp. 233–255, Mar. 2002.

[21] O. Nechushtan, B. Raveh, and D. Halperin, "Sampling-Diagrams Automata : a Tool for Analyzing Path Quality in Tree Planners," in *WAFR*, 2010.

[22] D. Ferguson and A. Stentz, "Anytime RRTs," in *IROS*, no. line 3. IEEE, Oct. 2006, pp. 5369–5375.

[23] R. Geraerts and M. H. Overamrs, "Creating small graphs for solving motion planning problems," in *MMAR*, 2005, pp. 531–536.

[24] R. Geraerts and M. H. Overmars, "Reachability-based analysis for probabilistic roadmap planners," *RAS*, vol. 55, pp. 824–836, 2007.

[25] M. A. Morales, R. Pearce, and N. M. Amato, "Metrics for analyzing the evolution of $\mathcal{C}$-space models," in *ICRA*, Orlando, FL, May 2006, pp. 1268–1273.

[26] J. Kim, R. A. Pearce, and N. M. Amato, "Extracting optimal paths from roadmaps for motion planning," in *ICRA*, vol. 2, 14-19 September 2003, pp. 2424–2429.

[27] D. Nieuwenhuisen and M. H. Overmars, "Using cycles in probabilistic roadmap graphs," in *ICRA*, 2004, pp. 446–452.

[28] R. Geraerts and M. H. Overmars, "Creating high-quality roadmaps for motion planning in virtual environments," in *IROS*, 2006, pp. 4355–4361.

[29] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime Motion Planning using the RRT," in *ICRA*, 2011.

[30] R. Pearce, M. Morales, and N. Amato, "Structural Improvement Filtering Strategy for PRM," in *RSS*, 2008.

[31] D. Peleg and A. Schäffer, "Graph Spanners," *Journal of Graph Theory*, vol. 13, no. 1, pp. 99–116, 1989.

[32] E. Cohen, "Fast Algorithms for Constructing t-Spanners and Paths with Stretch t," *Journal on Computing*, vol. 28, no. 1, p. 210, 1998.

[33] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, "Near-Linear Time Construction of Sparse Neighborhood Covers," *Journal on Computing*, vol. 28, no. 1, p. 263, 1998.

[34] M. Thorup and U. Zwick, "Approximate distance oracles," in *JACM*, vol. 52, no. 1. ACM, Jan. 2005, pp. 1–24.

[35] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares, "On sparse spanners of weighted graphs," *Discrete and Computational Geometry*, vol. 9, no. 1, pp. 81–100, 1993.

[36] L. Roditty, M. Thorup, and U. Zwick, “Deterministic constructions of approximate distance oracles and spanners,” in *ICALP*. Springer, 2005, pp. 261–272.

[37] S. Baswana and S. Sen, “A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs,” *Random Structures and Algorithms*, vol. 30, no. 4, pp. 532–563, July 2007.

[38] J. M. Keil, “Approximating the complete Euclidean graph,” in *Scandinavian Workshop on Algorithm Theory*. London, UK: Springer-Verlag, 1988, pp. 208–213.

[39] M. Elkin and D. Peleg, “Strong inapproximability of the basic k-spanner problem,” *Automata, Languages and Programming*, vol. 1853, pp. 636–648, 2000.

[40] I. A. Sucan, M. Moll, and L. E. Kavraki, “The open motion planning library (ompl),” http://ompl.kavrakilab.org, 2010.

[41] J. D. Marble and K. E. Bekris, “Asymptotically Near-Optimal is Good Enough for Motion Planning,” in *ISRR*, 2011.