Asymptotically Near-Optimal is Good Enough for Motion Planning

James D. Marble and Kostas E. Bekris

Abstract Asymptotically optimal motion planners guarantee that solutions approach optimal as more iterations are performed. There is a recently proposed roadmap-based method that provides this desirable property, the `PRM`* approach, which 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. From graph theory, there are many algorithms that produce sparse subgraphs, known as spanners, which can guarantee near-optimal paths. In this work, a method for interleaving an incremental graph spanner algorithm with the asymptotically optimal `PRM`* algorithm is described. The result 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. If smoothing is applied, the results are even more favorable for the near-optimal solution.

1 Introduction

Probabilistic roadmap planners [16] utilize an offline phase to build up knowledge about the configuration space (C-space) and solve many practical motion planning problems. Traditionally, many of these planners focus on feasibility and may return paths of low quality; considerably different from the optimal ones, where path quality can be measured in terms of length, clearance, or smoothness [34]. Smoothing can be used to improve some of these measures and algorithms exist that produce roadmaps with paths that are deformable to optimal ones [13, 31]. Hybridization graphs [29] 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, 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 eventually provide optimal solutions, 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, so they may not be feasible for some applications. The recently proposed `k-PRM`* algorithm [14] minimizes the

Computer Science and Engineering Department, University of Nevada, Reno 1664 N. Virginia St., MS 171, Reno, NV, 89557 e-mail: `{jmarble,bekris}@cse.unr.edu`

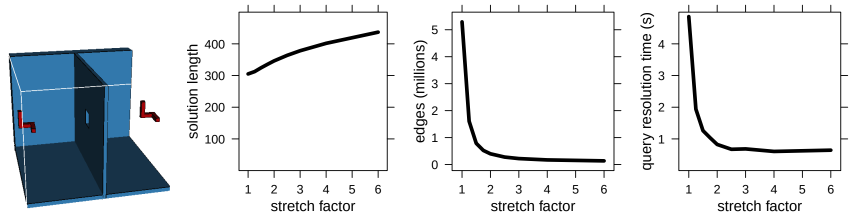

Fig. 1 Relaxing the requirement to find asymptotically optimal solutions, a large decrease in roadmap density and solution query time can be achieved for the problem shown on the left side. The larger this relaxation (stretch factor), the sparser the roadmap. These results are averaged over 1000 random queries on 10 runs of 50,000 vertex roadmaps. The `k-PRM*` algorithm corresponds to a stretch factor equal to 1, which results in a considerably higher query resolution time.

number of neighbors each new sample has to be connected to while still providing asymptotic optimality. Even so, the density of roadmaps produced by `k-PRM*` can be very high, resulting in slow online query resolution times, as shown in Figure 1.

This paper argues that a viable alternative is to compute roadmaps with asymptotically near-optimal guarantees. By relaxing the optimality guarantees, it is possible to construct roadmaps that are sparser, faster to build, and can answer queries more quickly while providing solution paths with near-optimal guarantees. Additionally, these roadmaps tend to return a solution in the same homotopic class as the optimum one, so smoothing brings path quality even closer to optimal in practice.

The theoretic foundations for this work lie in graph theory. In particular, graph spanners are sparse subgraphs with guarantees on path quality. Roadmaps with Useful Cycles [25] are, in fact, spanners. Edges that pass the "usefulness" test are added to the roadmap because not doing so would violate the guarantees about path quality.

In Section 3, this idea is applied to an asymptotically optimal roadmap planner to produce the Incremental Roadmap Spanner (`IRS`) algorithm. This planner can incrementally construct an asymptotically near-optimal roadmap faster than an asymptotically optimal one can be constructed. The resulting graph is also sparse, which is beneficial in any application where small roadmaps are preferable.

1.1 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 [2, 11, 28, 30, 32, 35].

Certain algorithms, such as the Visibility-based Roadmaps [32], the Incremental Map Generation algorithm [35] and the Reachability Roadmap Method [8] 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 [10].

A method is available to characterize the contribution of a sample to the exploration of the C-space [23], 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 high 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 [31], while Path Deformation Roadmaps compute paths that are deformable to optimal ones [13]. Another approach inspired by Dijkstra's algorithm extracts optimal paths from roadmaps for specific queries [18] but may require very dense roadmaps. The Useful Cycles approach implicitly creates a roadmap spanner with small number of edges [25] and has been combined with the Reachability Roadmap Method to construct connected roadmaps that cover 2D and 3D C-spaces and provide high quality paths [9]. A method has been introduced that filters nodes from a roadmap if they do not improve path quality measures [26].

Tree-based algorithms [12, 19] focus on single query planning, but they do not provide the preprocessing properties of roadmaps and are slower for multi-query applications. A tree is already a sparse graph and it becomes disconnected when removing edges. 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 [14, 24]. Anytime `RRT` [6] has been proposed as an approach to incrementally improve path quality.

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 can operate on such environments. These types of planners can benefit, however, 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 more quickly [4, 20].

The `RRG`, `RRT`*, and `PRM`* family of algorithms [14] provide asymptotic optimality for general configuration spaces. `RRG` and `RRT`* are based on `RRT`, a tree-based planner. The Anytime `RRT`* approach [15] extends `RRT`* with anytime planning in dynamic environments and can incrementally improve path quality. `PRM`* is a modification of standard `PRM`. The proposed technique is based on a variation of `PRM`*, i.e., `k-PRM`*, during the offline, roadmap-building step. More details on `k-PRM`* will be provided in Section 2.1.

1.2 Contribution

The contributions of this paper are the following:

1. The paper proposes a method for quickly constructing sparse roadmaps that provide high quality solutions to motion planning queries (`IRS`).
2. It provides an analysis of `IRS` showing the following:
   a. `IRS` has an asymptotic time complexity close to that of `k-PRM`*.
   b. Roadmaps constructed by `IRS` will provide solutions asymptotically near-optimal in the same spaces that `k-PRM`* would provide asymptotic optimality.

3. Experimental evidence showing that roadmaps constructed by `IRS` have the following properties:

    a. faster construction time
    b. sparsity, which results in faster online query resolution
    c. higher path quality than the theoretical lower bounds
    d. even more favorable comparison when smoothing is applied

Specifically, a method similar to Useful Cycles [25] is formalized as a graph spanner algorithm and is interleaved with `k-PRM`*. It is shown that a constant factor of the asymptotic optimality (asymptotic near-optimality), is maintained. `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` operates on an asymptotically optimal planner, it provides asymptotically near-optimal guarantees that the Useful Cycles approach did not since it employed a constant number of neighbors. `IRS` produces sparser roadmaps faster than `k-PRM`*. Previous work by the authors [22] has utilized state-of-the-art graph spanner algorithms with good complexity performance to prune the edges of a roadmap constructed with `k-PRM`*.

The proposed method balances 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 very poor solutions. On the other hand, the `k-PRM`* algorithm provides asymptotically optimal roadmaps that may 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, the stretch factor, that will return solutions arbitrarily close to the optimal ones, while still constructing sparse roadmaps relatively fast.

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, the majority of the edges can be removed while increasing mean path length by only a small amount, which can be further reduced by utilizing path 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 a marked decrease in construction time is also measured.

2 Foundations

A robot can be abstracted as a point in a d-dimensional configuration-space (C-space) where the set of collision-free configurations define $C_{\text{free}} \subset C$ [21]. The experiments performed for this paper use the space of rigid body configurations ($SE(3)$) as the C-space, but the proposed method is applicable to any C-space that is also a metric and probability space for reasons described in Section 3. 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 (The 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 $\sigma \in \Sigma = \{\rho | \rho : [0,1] \to C_{\text{free}}\}$ where $\sigma(0) = q_{\text{init}}$ and $\sigma(1) = q_{\text{goal}}$.

The `PRM` algorithm [16] can find solutions to the Path Planning Problem by sampling configurations in C_{free} then trying to connect them 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 the k-nearest neighbors are found. For each of these neighbors, an attempt is made to connect them to the sampled configuration with a local planner. If such a curve in C_{free} can be found, an edge between the two configurations is added to the roadmap.

Once the stopping criterion is met, the offline phase of the algorithm is complete. 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 is similar to the 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.

2.1 `k-PRM`*

Asymptotic optimality is the property of an algorithm that, given enough time, solutions to the path planning problem will almost surely converge to the optimum as defined by some 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 $c : \Sigma \to \mathbb{R}_{\geq 0}$ that admit a robust optimal solution with finite cost c^*, the probability that it will find a solution of cost c^* converges to 1 as the number of iterations approach infinity.

Asymptotic optimality is defined only for *robustly feasible* path planning problems as defined in the presentation of `k-PRM`* [14]. Such problems have a minimum clearance around the optimal solution and cost functions with a continuity property.

The standard `PRM`, as described above, attempts to connect sampled configurations to a fixed number, k, of nearest neighbors. It has been shown that this prevents the algorithm from providing asymptotic optimality [14]. The `k-PRM`* algorithm rectifies this deficiency by making k a logarithmic function of the size of the roadmap. Specifically, $k(n) = k_{\texttt{PRM}} \log n$ where $k_{\texttt{PRM}} > e(1 + 1/d)$. It has been proven that roadmaps constructed with this variation will almost surely converge to optimal solutions [14]. That is, `k-PRM`* is asymptotically optimal.

2.2 Graph Spanners

A graph spanner, as formalized in [27], 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.

A simple method for spanner construction, which is a generalization of Kruskal's algorithm for the minimum spanning tree, is given in Algorithm 1 [1]. Instead of accepting only edges that connect disconnected components, this algorithm accepts edges that provide shortcuts. Kruskal's algorithm is recovered by setting t to a large value.

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: return $G_S$
```

The inclusion criteria on line 1 ensures that no edges required for maintaining the spanner property are left out. From the global ordering of the edges performed on line 1, it has been shown that the number of edges retained by this algorithm is reduced from a potential $O(n^2)$ to $O(n)$ [1].

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 [3].

A number of spanner algorithms take advantage of the implicit weights of a Euclidean metric space to speed up the process. Most of these operate on *complete* Euclidean graphs [17, 7]. These algorithms can't be used on roadmaps that operate in C-spaces with obstacles because obstacles remove edges of the complete graph.

A concept central to the proposed technique and graph spanners is that of asymptotic *near*-optimality. This 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 (Asymptotic Near-Optimality in 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 approach infinity.

3 Approach

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 sequentially (first the roadmap, then the spanner), or incrementally by interleaving the individual steps of each task.

3.1 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. Simply applying an appropriate spanner algorithm to the roadmap accomplishes this and has been studied by the authors in previous work [22].

This approach can be wasteful, however. Every edge in G has been checked for collisions, but many of these will be discarded by the spanner algorithm. An incremental spanner algorithm that accepts or rejects on an edge-by-edge basis, such as Algorithm 1, is a better alternative. If all collision detection is deferred until after the offline phase of `k-PRM`* is complete (similar to lazy `PRM` [5]) then edges rejected by the spanner algorithm can skip collision detection all together. Since collision detection is frequently the most expensive operation in motion planning, the time savings could be substantial.

A potential downside to using an incremental spanner algorithm is higher time complexity. If the entire graph is available at once, then an algorithm that has linear time complexity can be used [3]. In practice, the time saved by avoiding collision detection frequently dominates the additional cost of using an incremental spanner algorithm.

3.2 Incremental Roadmap Spanner

The Incremental Roadmap Spanner (`IRS`, Algorithm 2) takes the idea of a sequential roadmap spanner 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 and before it is added to the roadmap.

Before discussing the implementation of the algorithm, some 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.

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`.

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

- a solution of sufficient quality has been found
- allotted time or memory have been exhausted
- 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 isn't 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 variation of Dijkstra's algorithm search can be employed. 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.

Algorithm 2 INCREMENTALROADMAPSPANNER(t)

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

First, the roadmap $G=(V,E)$ is initialized to empty (line 2). Then, it iterates until STOPPINGCRITERIA returns `true` (line 2). For each iteration, SAMPLEFREE is called and returns $v \in C_{\text{free}}$ (line 2). The number of nearest neighbors is calculated (line 2). The k-nearest neighbors of v, U are found by calling NEAR(V,v,k) (line 2). If NEAR does not return U ordered by distance from v, then it must be sorted on line 2. 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 2). Second, the edge must be checked for collision (line 2), as it is with other variations of PRM. 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 2). Finally, the sampled vertex is added to the roadmap (line 2) and 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 in Section 3.3. A *local* ordering of potential edges is performed on line 2. This doesn't 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, as shown in Section 4.

3.3 Analysis

Theorem 1. `IRS` *is asymptotically near-optimal.*

Proof. The proof of Theorem 1 relies on the asymptotic optimality of `k-PRM`* [14], on which `IRS` is based. The proposed technique has two differences from `k-PRM`*.

First, on line 2, the potential neighbors of a newly added vertex are ordered by non-decreasing 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, on line 2, 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 such roadmaps, $G = (V, E)$ returned by `k-PRM`* and $G_S = (V, E_S \subset E)$ returned by `IRS`. For each rejected 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 2.

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. □

3.4 Time Complexity

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

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. So, the total running time of `k-PRM`* is $O\big(n \cdot (\log n + \log n \cdot \log^d p)\big)$, which simplifies to $O(n \cdot \log n \cdot \log^d p)$.

- Time complexity breakdown for `IRS` (ignoring constant time operations):
  - For each of n iterations:
    - NEAR (ε-approximate): $\log n$
    - neighbor ordering: $\log n \cdot \log\log n$

- For each $\log n$ neighbors:
    - COLLISIONFREE: $\log^d p$
    - 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 distance to the sampled vertex. Because many implementations of NEAR return the list of neighbors in order of distance, the cost of this 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 that. 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

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

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

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

which is asymptotically slower than k-PRM*. However, as will be shown experimentally in Section 4, the very large constants involved in collision checking that can be skipped by IRS makes this a faster algorithm for roadmaps of practical size.

3.5 Size Complexity

The number of edges in a spanner produced by Algorithm 1 is $O(n)$ with constants related to t and d. This bound does not hold for IRS because, 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.

4 Evaluation

All experiments were run using the Open Motion Planning Library (OMPL) [33] on 2GHz processors with 4GB of memory. Four representative environments were chosen from those distributed with OMPL in addition to the simple environment shown in Fig. 1.

alpha puzzle: The entire free space is highly constrained in this classical motion planning benchmark (Fig. 2(a)).

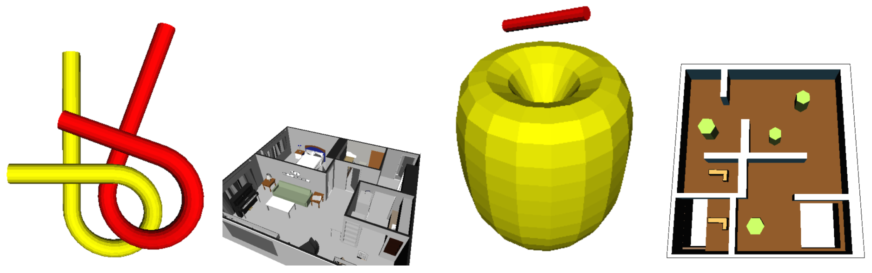

Fig. 2 Environments used in the experiments with example configurations for the robot.

apartment: The piano is the robot in this environment with a very expensive collision detection cost (Fig. 2(b)).

bug trap: A rod shaped robot must orient itself to fit through the narrow passage and escape a three-dimensional version of the classical bug trap (Fig. 2(c)).

cubicles: Two floors of loosely constraining walls and obstacles must be navigated (Fig. 2(d)).

For each environment, five roadmaps of 50,000 vertices were generated by `k-PRM`, `k-PRM`*. For `IRS`, 10 different roadmaps were generated for each of 8 stretch factors. On each of these roadmaps, 1000 random start and goal pairs were queried and various qualities of the resulting solutions were measured.

4.1 Construction Time

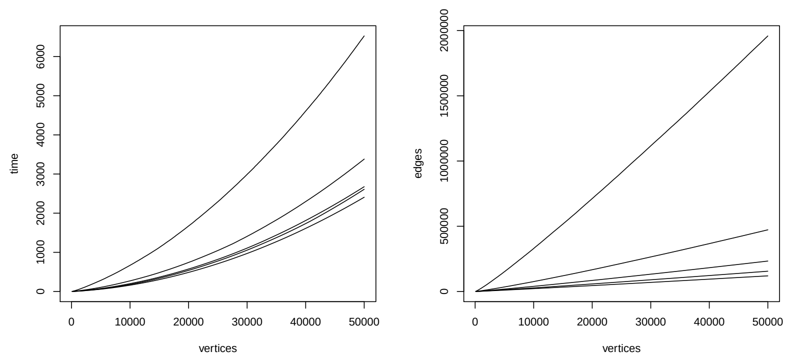

Fig. 3 Construction time and roadmap density for `k-PRM`* and `IRS` in the apartment environment averaged over 5 runs. Despite a higher asymptotic complexity, practically sized roadmap spanners can be more quickly constructed because fewer edges must be checked for collision. A dramatic reduction in the number of edges is shown for stretch factor as low as 1.5.

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 these experiments. Since `IRS` reduces the number of collision checks required, running time is reduced the higher the stretch factor is. This is shown in Fig. 3, where a stretch factor of $t = 2$ allows `IRS` to construct a 50,000 node roadmap in under half 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.

4.2 Space Requirements

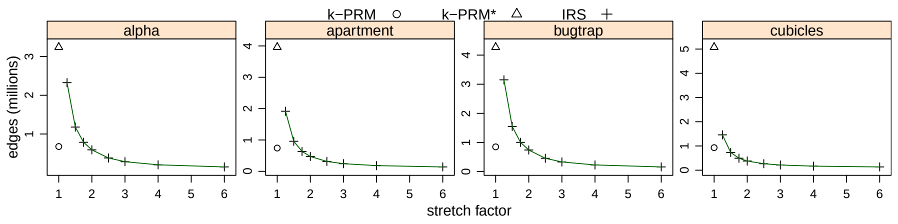

Fig. 4 The roadmap density in 10 different 50,000 node roadmaps for each environment. Environments with denser obstacles have fewer edges than `k-PRM`*, but gain a smaller improvement from `IRS`.

While each roadmap contains the same number of vertices (50,000), the space required for connectivity information is reduced by up to 95%. 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.

4.3 Solution Quality

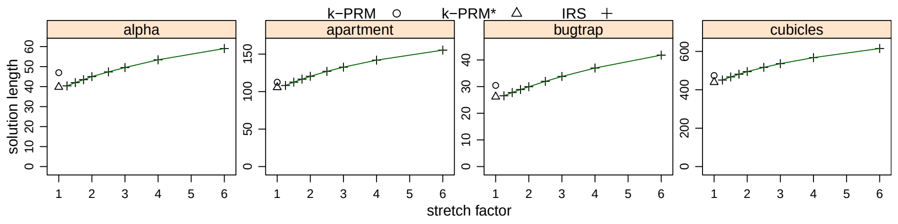

Fig. 5 Mean solution length of 1000 random query pairs on 10 different 50,000 node roadmaps for each environment. Solution length increases when the stretch factor is increased, but the mean increase is much less than the upper bound guaranteed by the algorithm.

In Fig. 5, path quality is measured by querying a roadmap with 1000 random start and goal configurations. The lengths of the resulting paths increase as the number of edges in the spanner is reduced. For these random starting and goal configurations, the average extra cost is much shorter than the worst case guaranteed by the stretch factor.

It is interesting which paths are most affected by this increase in path length. 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. 6 as a function of its length in a roadmap generated by `k-PRM`*. All shortest paths to 10 random vertices are averaged over 5 different roadmaps.

In Fig. 7, the trade-off between roadmap density and path quality is directly compared. Although `k-PRM` can provide either a sparser roadmap or higher quality solutions than `IRS`, it cannot provide a better trade-off except in the apartment environment.

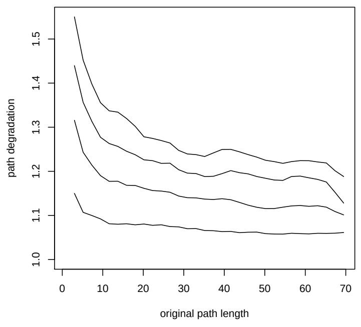

Fig. 6 Mean path length degradation for different stretch factors as a function of the path length in the original roadmap.

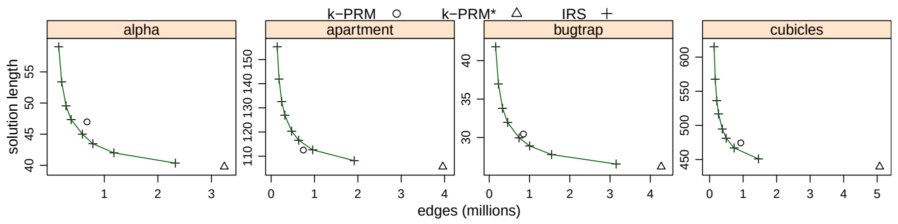

Fig. 7 Trade-off between solution length and roadmap density averaged over 1000 random query pairs on 10 different 50,000 node roadmaps for each environment. Each data point for `IRS` represents a different stretch factor from 1.5 to 6.

4.4 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 affected by the number of edges in the roadmap, only the number of vertices. Since each roadmap has the same number of vertices, the roadmap connection time is fixed and variations in query resolution time can be attributed to differences in the running time of the A* search. As shown in Fig. 8, this variation is very large. Removing edges from the roadmaps reduces the query resolution time by up to 70%.

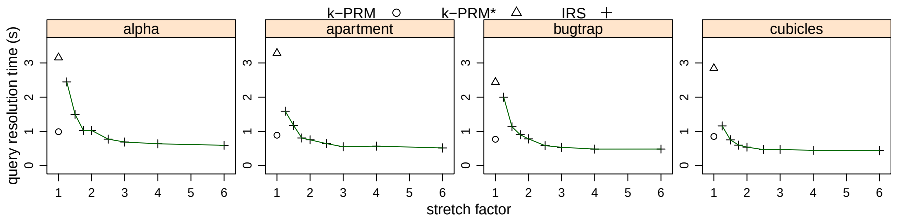

Fig. 8 Mean query resolution time of 1000 random query pairs on 10 different 50,000 node roadmaps for each environment. The online portion of the algorithms is reported by this chart and includes the time it takes to connect the start and goal configurations as well as the A* graph search. Higher stretch factors produce roadmaps with fewer edges that can be searched more quickly.

4.5 Effects of Smoothing

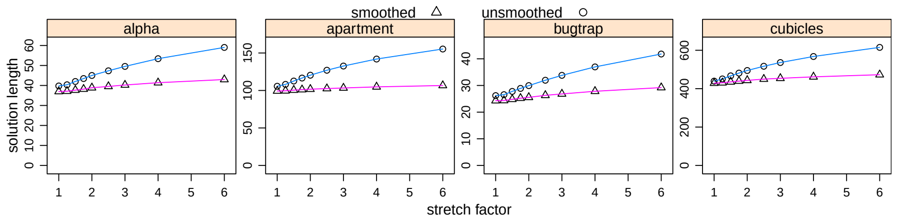

Fig. 9 A comparison between smoothed and unsmoothed solution lengths of 1000 random query pairs on 10 different 50,000 node roadmaps for each environment. The relative solution length increase for higher stretch factors is lower for the smoothed solutions. This reflects the ability of spanners to preserve homotopic path classes. Note that a stretch factor of $t = 1$ corresponds to `k-PRM*`.

For each query, a simple approach for path smoothing was tested. 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, but it executes in less than 40 milliseconds, and produced impressive results (Fig. 9). Note that the gap between the smoothed and unsmoothed solution length indicates that both `k-PRM*` and `IRS` have yet to converge to the optimal solution.

The smoothing time increases as the sparsity of the roadmap increases. This reflects the larger number of vertices in solutions from these roadmaps. However, the time taken to smooth the solutions is two 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.

5 Discussion

This work shows that it is practical to compute sparse roadmaps in 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 stretch factor parameter provides the ability to tune this trade-off. The experiments confirm that roadmaps constructed with lower stretch factors can have higher path quality but are denser.

The existing approach removes only edges from the original roadmap. Since, however, the roadmap is embedded in a continuous 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 that the quality of a path answering a query in the continuous space is guaranteed not to get worse than the specified stretch factor.

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 C-space [13]. 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. Althöfer, I., Das, G., Dobkin, D., Joseph, D., Soares, J.: On sparse spanners of weighted graphs. Discrete and Computational Geometry **9**(1), 81–100 (1993)
2. Amato, N.M., Bayazit, O.B., Dale, L.K., Jones, C., Vallejo, D.: OBPRM: An Obstacle-based PRM for 3D Workspaces. In: Workshop on the Algorithmic Foundations of Robotics (WAFR), pp. 155–168 (1998)
3. Baswana, S., Sen, S.: A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. Random Structures and Algorithms **30**(4), 532–563 (2007)
4. Bekris, K., Kavraki, L.: Informed and Probabilistically Complete Search for Motion Planning under Differential Constraints. In: First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR). Chicago, IL (2008)
5. Bohlin, R., Kavraki, L.: Path Planning Using Lazy PRM. In: IEEE Intl. Conf. on Robotics and Automation (ICRA), vol. 1, pp. 521–528. San Francisco, CA (2000)
6. Ferguson, D., Stentz, A.: Anytime RRTs. In: IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), pp. 5369–5375. Beijing, China (2006)
7. Gao, J., Guibas, L.J., Nguyen, A.: Deformable spanners and applications. Computational geometry : theory and applications **35**(1-2), 2–19 (2006)
8. Geraerts, R., Overamrs, M.H.: Creating Small Graphs for Solving Motion Planning Problems. In: IEEE International Conference on Methods and Models in Automation and Robotics (MMAR), pp. 531–536 (2005)
9. Geraerts, R., Overmars, M.H.: Creating High-Quality Roadmaps for Motion Planning in Virtual Environments. In: IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), pp. 4355–4361. Beijing, China (2006)
10. Geraerts, R., Overmars, M.H.: Reachability-based Analysis for Probabilistic Roadmap Planners. Journal of Robotics and Autonomous Systems (RAS) **55**, 824–836 (2007)
11. Guibas, L.J., Holleman, C., Kavraki, L.E.: A Probabilistic Roadmap Planner for Flexible Objects with a Workspace Medial-Axis-Based Sampling Approach. In: IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), vol. 1, pp. 254–259 (1999)

12. Hsu, D., Kindel, R., Latombe, J.C., Rock, S.: Randomized Kinodynamic Motion Planning with Moving Obstacles. International Journal of Robotics Research **21**(3), 233–255 (2002)
13. Jaillet, L., Simeon, T.: Path Deformation Roadmaps. In: Workshop on the Algorithmic Foundations of Robotics (WAFR). New York City, NY (2006)
14. Karaman, S., Frazzoli, E.: Sampling-based algorothms for optimal motion planning. The International Journal of Robotics Research **30**(7), 846–894 (2011)
15. Karaman, S., Walter, M., Perez, A., Frazzoli, E., Teller, S.: Anytime Motion Planning using the RRT. In: IEEE Intl. Conf. on Robotics and Automation (ICRA). Shanghai, China (2011)
16. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation (TRA) **12**(4), 566–580 (1996)
17. Keil, J.M.: Approximating the complete Euclidean graph. In: 1st Scandinavian workshop on algorithm theory, pp. 208–213. Springer-Verlag, London, UK (1988)
18. Kim, J., Pearce, R.A., Amato, N.M.: Extracting Optimal Paths from Roadmaps for Motion Planning. In: IEEE Intl. Conf. on Robotics and Automation (ICRA), vol. 2, pp. 2424–2429. Taipei, Taiwan (2003)
19. LaValle, S.M., Kuffner, J.J.: Randomized Kinodynamic Planning. International Journal of Robotics Research **20**(5), 378–400 (2001)
20. Li, Y., Bekris, K.E.: Learning Approximate Cost-to-Go Metrics To Improve Sampling-based Motion Planning. In: IEEE Intl. Conf. on Robotics and Automation (ICRA). Shanghai, China (2011)
21. Lozano-Perez, T.: Spatial planning: A configuration space approach. IEEE Transactions on Computers pp. 108–120 (1983)
22. Marble, J.D., Bekris, K.E.: Computing Spanners of Asympotically Optimal Probabilistic Roadmaps. In: IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS). San Francisco, CA (2011)
23. Morales, M.A., Pearce, R., Amato, N.M.: Metrics for Analyzing the Evolution of C-Space Models. In: IEEE Intl. Conf. on Robotics and Automation (ICRA), pp. 1268–1273 (2006)
24. Nechushtan, O., Raveh, B., Halperin, D.: Sampling-Diagrams Automata : a Tool for Analyzing Path Quality in Tree Planners. In: Workshop on the Algorithmic Foundations of Robotics (WAFR). Singapore (2010)
25. Nieuwenhuisen, D., Overmars, M.H.: Using Cycles in Probabilistic Roadmap Graphs. In: IEEE Intl. Conf. on Robotics and Automation (ICRA), pp. 446–452 (2004)
26. Pearce, R., Morales, M., Amato, N.: Structural Improvement Filtering Strategy for PRM. In: Robotics: Science and Systems (RSS). Zurich, Switzerland (2008)
27. Peleg, D., Schäffer, A.: Graph Spanners. Journal of graph theory **13**(1), 99–116 (1989)
28. Plaku, E., Bekris, K.E., Chen, B.Y., Ladd, A.M., Kavraki, L.E.: Sampling-Based Roadmap of Trees for Parallel Motion Planning. IEEE Transactions on Robotics and Automation (TRA) **21**(4), 587–608 (2005)
29. Raveh, B., Enosh, A., Halperin, D.: A Little More, a Lot Better: Improving Path Quality by a Path-Merging Algorithm. IEEE Transactions on Robotics **27**(2), 365–370 (2011)
30. Sanchez, G., Latombe, J.C.: A Single-Query, Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking. In: International Symposium on Robotics Research, pp. 403–418 (2001)
31. Schmitzberger, E., Bouchet, J.L., Dufaut, M., Wolf, D., Husson, R.: Capture of Homotopy Classes with Probabilistic Roadmap. In: IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), pp. 2317–2322. Switzerland (2002)
32. Simeon, T., Laumond, J.P., Nissoux, C.: Visibility-based probabilistic roadmaps for motion planning. Advanced Robotics Journal **41**(6), 477–494 (2000)
33. Sucan, I.A., Moll, M., Kavraki, L.E.: The Open Motion Planning Library (OMPL). `http://ompl.kavrakilab.org` (2010)
34. Wein, R., van den Berg, J., Halperin, D.: Planning High-Quality Paths and Corridors amidst Obstacles. The International Journal of Robotics Research **27**(11-12), 1213–1231 (2008)
35. Xie, D., Morales, M., Pearce, R., Thomas, S., Lien, J.L., Amato, N.M.: Incremental Map Generation (IMG). In: Workshop on Algorithmic Foundations of Robotics (WAFR). New York City, NY (2006)