# Sparse Roadmap Spanners for Asymptotically Near-Optimal Motion Planning

**Andrew Dobson, Kostas E. Bekris**

**Abstract**

*Asymptotically optimal planners, such as* PRM$^*$*, guarantee that solutions approach optimal as the number of iterations increases. Roadmaps with this property, however, may grow too large for storing on resource-constrained robots and for achieving efficient online query resolution. By relaxing optimality, asymptotically near-optimal planners produce sparser graphs by not including all edges. The idea stems from graph spanners, which produce sparse subgraphs that guarantee near-optimality. Existing asymptotically optimal and near-optimal planners, however, include all sampled configurations as roadmap nodes, meaning only infinite-size graphs have the desired properties. To address this limitation, this work describes* SPARS*, an algorithm that returns a sparse roadmap spanner. The method provides the following properties: (a) probabilistic completeness, (b) asymptotic near-optimality and (c) the probability of adding nodes to the spanner converges to zero as iterations increase. The last point suggests that finite-size data structures with asymptotic near-optimality in continuous spaces may indeed exist. The approach builds simultaneously a dense graph similar to* PRM$^*$ *and its roadmap spanner, meaning that upon construction an infinite-size graph is still needed asymptotically. An extension of* SPARS *is also presented, termed* SPARS2*, which removes the dependency on building a dense graph for constructing the sparse roadmap spanner and for which it is shown that the same desirable properties hold. Simulations for rigid body motion planning show that algorithms for constructing sparse roadmap spanners indeed provide small data structures and result in faster query resolution. The rate of node addition is shown to decrease over time and practically the quality of solutions is considerably better than the theoretical bounds. Upon construction, the memory requirements of* SPARS2 *are significantly smaller but there is a small sacrifice in the size of the final spanner relative to* SPARS*.*

**Keywords**

sampling-based motion planning, asymptotic optimality, near-optimality, roadmap spanners

## 1 Introduction

An important motion planning challenge is the identification of compact graphical representations for answering shortest-path queries in configuration spaces ($C$-spaces) of moving systems (Latombe, 1991; Agarwal, 2011). There is a broad range of applications, which can benefit from compact data structures that provide high-quality paths. One example is cloud-enabled, resource-constrained robots, where preprocessing can be performed offline on a computing cloud and the final planning structure is communicated wirelessly to individual robots (Kuffner, 2010). The resulting structure should be small so as to minimize strain on the communication network and the robot. Planning among dynamic obstacles can also utilize such planning structures to quickly recompute paths once old ones have been invalidated by a moving obstacle (Kallman and Mataric, 2004). Another application is in the game industry, where small computational budgets are given to path planning, necessitating a structure which is queried quickly and returns high-quality paths (Geraerts and Overmars, 2006). This work describes how to generate such compact data-structures, which return high-quality paths. The objective is to show that these structures are practical and can be constructed in a time and memory efficient manner, while it is also possible to prove desirable theoretical guarantees.

Sampling-based motion planners (Choset et al., 2005; LaValle, 2006), and especially roadmap methods, such as the Probabilistic Roadmap Method (PRM) (Kavraki et al., 1996), are particularly appropriate in this context. These methods build during a preprocessing phase a discrete

Computer Science Department, Rutgers University, NJ, USA.

**Corresponding author:**
Kostas Bekris, Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ, 08854, USA.
Email: kostas.bekris at cs.rutgers.edu

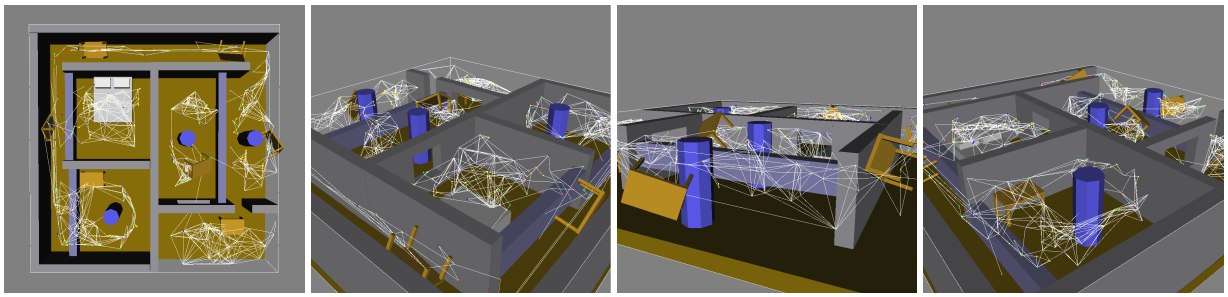

Figure 1: A roadmap spanner in the $SE(3)$ "Beam Site" environment, loaded in the OMPL software package (Şucan et al., 2012). Configurations for a table moving along a solution path are highlighted.

representation of a $C$-space, in the form of a roadmap, that can then be used to answer motion planning queries. To achieve online efficiency, they should return a small, sparse roadmap that can be queried quickly, and returns paths which are close to optimal. In general continuous spaces, however, only asymptotic optimality can be provided, where the solution converges to the optimum as more preprocessing time is provided. A recent result describes the conditions under which sampling-based planners, such as PRM$^*$ (Karaman and Frazzoli, 2011), converge to asymptotically optimal solutions. While these methods provide very desirable properties regarding path quality, the resulting graph is not compact and can get very large to store on a resource-constrained robot or to efficiently query on the fly. Moreover, it is not clear when to stop the sampling process.

Recent efforts have attempted to address this issue by relaxing optimality requirements (Marble and Bekris, 2011a) and drawing inspiration from work on graph spanners (Peleg and Schäffer, 1989). Spanners are subgraphs where the shortest path between two nodes is no longer than $t$ times the shortest path on the original graph. The parameter $t$ is called the stretch factor of the spanner. An incremental integration of asymptotically optimal roadmaps with graph spanners results in efficient roadmap spanners that provide asymptotic near-optimality (Marble and Bekris, 2011b). Spanners, however, only remove edges, and the resulting roadmaps, similar to asymptotically optimal solutions, still need to include every $C$-space sample as a node to achieve this property.

The current contribution extends this line of work and develops practical methods for computing sparse roadmap spanners with asymptotic near-optimality guarantees through a $C$-space sampling process. The resulting structure is sparse, in the sense that it does not include the vast majority of $C$-space points as nodes and also includes a small number of edges relative to asymptotically optimal roadmaps. Note that the notion of sparsity in this work deviates from its standard use in graph theory (linear number of edges as a function of the number of nodes). The proposed planners identify appropriate criteria for selecting which samples are needed in the sparse roadmap spanner while upholding theoretical guarantees regarding path quality. The asymptotic near-optimality guarantees are appropriately adapted relative to previous contributions to include an additive term in order to achieve the desired sparsity in the size of the data structure. In particular, the planners for constructing sparse roadmap spanners presented in this work have the following properties:
(i) probabilistic completeness,
(ii) asymptotic near-optimality with additive cost, i.e., the resulting roadmap can answer any path planning query in the $C$-space with paths of length:

$$t \cdot c^* + 4 \cdot \Delta, \tag{1}$$

where $t$ and $\Delta$ are input parameters to the algorithm, $c^*$ is the cost of the optimum path, if one exists, between the query points in $C_{\text{free}}$ with clearance at least $cl$ and
(iii) asymptotic sparsity, i.e., the probability of adding new nodes and edges converges to 0 as the number of iterations increases.

This framework is a candidate solution to the problem of finding a compact representation for answering shortest-path queries in continuous spaces. To the best of the author's knowledge, no previous line of work argues about the existence of finite data structures with some form of near-optimality guarantee for continuous path planning. This work provides an indication that such data structures can be created. Furthermore, an important advantage of this framework is that it gives rise to a natural stopping criterion inspired by the Visibility − based PRM approach (Simeon et al., 2000). The criterion relates to a probabilistic measure of how close the roadmap is to a solution that provides the desired properties.

This paper first describes the overall framework for

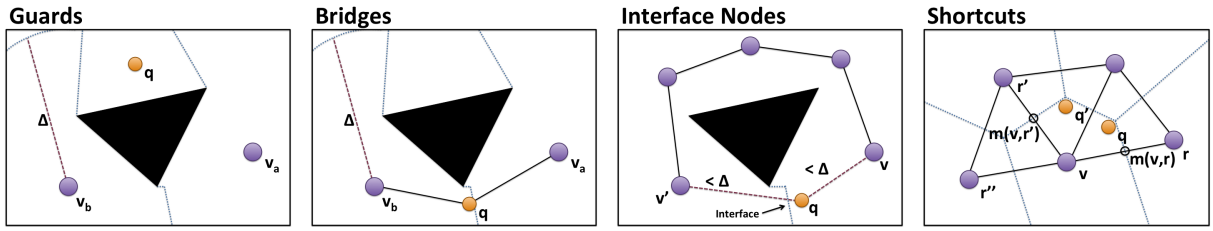


Figure 2: The four types of samples $\rho \in C$-space that sparse roadmap spanners are considering for addition. Nodes may be added for coverage (guards), connectivity (bridges), or to connect nodes which share an "interface" (interface nodes), or to satisfy near-optimality constraints when efficient $C$-space paths ($\rho$ to $\rho'$) are found (shortcuts).

building sparse roadmap spanners. Figure 2 shows the conditions under which configurations are included in the spanner. It has to be they are useful for coverage or connectivity or that they improve the quality of paths on the sparse roadmap spanner relative to optimum paths in the obstacle-free $C$-space. Given this framework, the paper then introduces a concrete implementation, called SPARS, that constructs in parallel to the spanner an asymptotically optimal dense graph using PRM* that includes all $C$-space samples. The overall approach deviates from a straightforward integration of PRM* with graph spanners. Instead it follows a novel methodology that is based on the identification of boundaries of "visibility" regions for the spanner nodes by utilizing their connectivity with dense graph nodes. It eventually guarantees that all shortest paths between boundaries of "visibility" regions on the dense graph can be represented by paths on the spanner that are at most $t$ times longer, which leads to Eq. 1. The parameters $t$ and $\Delta$ control the sparsity of the spanner.

SPARS lends itself to an analysis that shows that it provides the desirable theoretical properties. Nevertheless, the use of the dense graph is a significant limitation. So, while the method returns a sparse roadmap spanner with the desirable properties, its computational requirements are still significant during construction, especially in terms of memory. For this reason, a second approach, SPARS2, is also presented that extends the previous one and is shown to provide the same theoretical guarantees without explicitly maintaining the dense graph. In SPARS2, the process for adding samples to the sparse roadmap spanner for coverage or connectivity is the same as it does not depend on the dense graph. On the other hand, "shortcut" nodes do require $C$-space shortest path information. This work shows that it is possible to relax the requirements for adding such nodes, i.e., add more nodes than needed for the desired near-optimality guarantees, so that the dense graph is not needed. The method introduces some conservative approximations of shortest paths in the $C$-space, computed through a local sampling process and bookkeeping information. The space requirements of this bookkeeping is a finite function of the number of spanner nodes. Since the probability of adding new nodes converges to zero, the probability of increasing the bookkeeping data also goes to zero.

Simulations were performed on the Open Motion Planning Library (OMPL) (Şucan et al., 2012) to evaluate the proposed framework and its two instantiations, SPARS and SPARS2, against the asymptotically optimal PRM*. The experiments indicate that, during construction, SPARS2 is using a significantly smaller amount of memory. The returned spanner has similar sparsity characteristics as the one returned by SPARS, alas with a small sacrifice in size (i.e., SPARS returns a smaller sparse roadmap spanner than SPARS2, since it has access to the dense graph and doesn't follow as conservative approximations for shortest paths in the $C$-space). For both algorithms, the resulting graph is orders of magnitude smaller than the asymptotically optimal roadmap for the same construction time. Interestingly, paths returned by the spanners are very competitive to those computed by PRM* for the same amount of construction time and significantly better than the theoretical guarantees. Online query resolution times are very efficient on sparse roadmap spanners and are significantly shorter than query resolution on the output of PRM* for competitive path quality. Furthermore, the rate of nodes added to the spanner does decrease over time.

The next section relates this work to the state-of-the-art and is followed by a presentation of the problem statement and a helpful set of formal definitions and notation. Then, the paper presents the framework for constructing sparse roadmap spanners and its theoretical analysis. The experimental results validate the properties and show the practicality of the framework and especially of SPARS2. The paper concludes with a discussion of the methods and future research directions.

## 2 Background

**Piano Mover's Problem** The general mover's problem is posed for a collection of polyhedral rigid bodies in a polyhedral workspace and is PSPACE − HARD (Reif, 1979). A PSPACE algorithm introduced the roadmap, a 1-D subspace of $C$-space that captures its connectivity (Canny, 1988). The algorithm, however, is impractical, similarly to $C$-space approximations (Brooks and Lozano-Pérez, 1983; Lozano-Pérez, 1983; Kambhampati and Davis, 1986). A paradigm based on potential fields was developed (Khatib, 1986; Hwang and Ahuja, 1992), but complete versions of this methodology (Koditschek, 1989; Rimon and Koditschek, 1992) are difficult to apply in general configuration spaces. A similar technique that solved difficult problems took a stochastic approach to avoid local minima and motivated the development of sampling-based planners (Barraquand and Latombe, 1991; Lamiraux and Laumond, 1996).

**Probabilistic Roadmaps:** The first popular method for employing a sampling strategy in order to construct a roadmap in the $C$-space was the Probabilistic Roadmap Method (PRM) (Kavraki et al., 1996). The algorithm samples a configuration in $C_{\text{free}}$, the collision-free $C$-space, and adds it as a node to the roadmap. It then tries to connect it with a local path to its $k$-closest neighbors among the existing nodes ($k$-PRM) or those within a $\delta$-ball ($\delta$-PRM). If this local path, typically a straight line in the $C$-space, is in $C_{\text{free}}$, an edge is added to the roadmap. The advantages of the method lie on its simplicity, elegance and generality. While path non-existence cannot be proved with the PRM, probabilistic completeness can be provided instead: if a path exists, it will be found eventually. This was originally proven for $k$-dimensional manifolds (Kavraki et al., 1998; Hsu et al., 1998), for nonholonomic robots (Švestka, 1997) and later for a broad class of problems (Ladd and Kavraki, 2004).

**Variations of PRM:** The efficiency of probabilistic roadmaps is dominated by the presence of "narrow passages", which require sampling from a very small set to solve them. This motivated the development of variations that appropriately sampled configurations to speed up the construction of sufficiently connected roadmaps (Amato et al., 1998; Hsu et al., 1998; Boor et al., 1999; Wilmarth et al., 1999; Guibas et al., 1999; Bohlin and Kavraki, 2000; Foskey et al., 2001; Hsu et al., 2003; Leven and Hutchinson, 2003; Plaku et al., 2005). A number of quasi-random alternatives to random sampling have also been proposed (Branicky et al., 2001) and there has been a comparative study on the subject (Geraerts and Overmars, 2003). Furthermore, the PRM framework has been adapted so as to solve a variety of different challenges beyond the basic piano mover's problem involving multiple robots (Sánchez and Latombe, 2002), manipulation planning (Nielsen and Kavraki, 2000), assembly planning (Sundaram et al., 2001), planning for flexible objects (Lamiraux and Kavraki, 2001) and bioinformatics applications (Apaydin et al., 2003).

**Visibility-based PRM:** A variation of PRM closely related to the current work is the Visibility − based PRM (Simeon et al., 2000). This method focuses on rejecting specific samples while adding only those required for coverage and connectivity purposes. The combination of these two properties is sufficient for probabilistic completeness. The resulting roadmaps are small-size tree structures, because two nodes are not connected if they belong to the same connected component. Furthermore, the technique provides an automatic stopping criterion: when $M$ consecutive samples fail to be added to the roadmap, then a probabilistic estimation of the percentage of free space not covered by the nodes of the data structure is $\frac{1}{M}$.

**Variations Focusing on Path Quality** An issue with PRM variants that employed the connected component heuristic to speed up roadmap construction is that the paths returned can often be of low quality. This led to work that aimed to identify "useful cycles", which adds edges between roadmap nodes if the existing path connecting the nodes is sufficiently lengthy (Nieuwenhuisen and Overmars, 2004). The resulting structure is no longer a tree, but remains sparse and improves path quality. The "useful cycles" criterion has been combined with the Reachability Roadmap variant of PRM to return high clearance paths in 2D and 3D $C$-spaces (Geraerts and Overmars, 2006). One way to improve path quality is through a post-processing, smoothing phase. Methods that reason about path homotopy provide solutions that can be smoothed to optimal ones (Jaillet and Simeon, 2006; Schmitzberger et al., 2002). Hybridization graphs can be seen as a smoothing process that combines multiple solutions into a single, better quality one (Raveh et al., 2011). While such smoothing-based approaches are valid alternatives in certain cases, they construct relatively dense roadmaps and increase the online query resolution time.

**Tree Sampling-based Planners:** Alternatives to roadmap-based methods explore the $C$-space using a tree data structure by incrementally propagating paths from existing configurations, such as the popular Rapidly-exploring Random Tree (RRT) approach (LaValle and Kuffner, 2001). They tend to be more efficient in quickly answering individual queries and they can be easily applied to problems involving dynamics because they do not

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

Table 1: PRM variations and asymptotic optimality properties. Parameter $n$: # nodes in the roadmap.

depend on the existence of a steering method that exactly connects two states of the system. Furthermore, they already return sparse data structures. Nevertheless, they do not provide the same properties in terms of preprocessing the entire $C$-space in order to be able to answer multiple, unknown queries. Moreover, the basic RRT approach has been shown to almost certainly converge to suboptimal solutions (Nechushtan et al., 2010). It can be extended, however, to an asymptotically optimal variant, known as RRT$^*$ (Karaman and Frazzoli, 2010). Even tree-based methods can benefit from sparse roadmaps that can quickly return C-space distances among obstacles (Li and Bekris, 2011).

**Conditions for Asymptotic Optimality:** Important recent work has provided the conditions under which sampling-based methods are asymptotically optimal (Karaman and Frazzoli, 2010, 2011). Asymptotic optimality implies that the quality of solutions converges to optimal as time goes to infinity. The analysis indicates that for the PRM the important variable is the number of neighbors that each new sample should be connected to, as indicated in Table 1. A simple PRM that connects samples to neighbors within a $\delta$-ball is asymptotically optimal, but results in a dense roadmap. The roadmap's density can be reduced by considering the $k$-nearest neighbors of the sample, but this version is not asymptotically optimal. PRM$^*$ and $k-$PRM$^*$ rectify this by selecting the minimum number of neighbors required for asymptotic optimality, which is a logarithmic function of the number of nodes (Karaman and Frazzoli, 2010, 2011). Nevertheless, all samples are added as nodes, resulting in a large graph and the resulting structure is still relatively dense. Thus, roadmaps with asymptotic optimality properties have large memory requirements and take longer to query than other approaches, which focus on sparsity. Furthermore, it is not clear when to stop sampling.

**Asymptotic Near-Optimality:** A way to return sparser, good-quality roadmaps is to relax the optimality guarantees by utilizing graph spanners (Peleg and Schäffer, 1989). Spanners are subgraphs, where the shortest path between two nodes on the subgraph is no longer than $t$ times the shortest path on the original graph, where $t$ is the stretch factor of the spanner. Applying an efficient spanner (Baswana and Sen, 2007) on the output of k $-$ PRM$^*$ resulted in a Sequential Roadmap Spanner (SRS) (Marble and Bekris, 2011a), which reduces the expected number of edges and provides asymptotic near-optimality, i.e., as more time is spent on constructing the roadmap, the quality of solutions converges to a value at most $t$ times the optimal. An incremental integration of spanners with k $-$ PRM$^*$ (IRS) has been experimentally shown to provide even better results (Marble and Bekris, 2011b). The path quality degradation with these methods is quite smaller in practice than the theoretical guarantees.

**Sparse Roadmaps Spanners:** The drawback of asymptotically optimal planners and of near-optimal spanners is that they include every sample. Thus, they need an infinite number of nodes to achieve their properties. An initial attempt towards not including all nodes was a simple extension of the IRS approach but did not provide any theoretical guarantees (Marble and Bekris, 2012). SPARS is the first method to the authors' knowledge that does provide for asymptotic near-optimality and where the probability of adding new nodes to the roadmap goes to zero (Dobson et al., 2012). SPARS2 achieves the same objectives while reducing the memory requirements upon construction (Dobson and Bekris, 2013 - submitted). The current manuscript brings together the last two contributions in a common framework for constructing sparse roadmap spanners. It also presents an extended analysis of their properties and an extensive evaluation.

# 3 Problem Setup and Terminology

The configuration space ($C$-space) abstraction is the set of configurations, where a point q in this space fully describes the volume occupied by a moving system. The $C$-space can be partitioned into two sets, one representing the collision-free part ($C_{\text{free}}$) and the subset which collides with obstacles. This work aims to solve the Path Planning Problem defined below by providing a discrete, graphical representation of the free space which can be queried effectively upon demand. The results presented here are for planning problems involving rigid bodies ($SE(2)$ and $SE(3)$). Nevertheless, the method should be applicable in any space where an appropriate metric and sampling function exist.

**Defn. 1 (The Path Planning Problem)** *Given the set of free configurations $C_{free} \subset C$-space, initial and goal configurations $\mathtt{q}_{init}, \mathtt{q}_{goal} \in C_{free}$, find a continuous path $\pi \in \Pi = \{\mathtt{q} | \mathtt{q} : [0,1] \rightarrow C_{free}\}$, $\pi(0) = \mathtt{q}_{init}$ and $\pi(1) = \mathtt{q}_{goal}$.*

The focus of this work, as with previous work that deals with asymptotic (near-)optimality, is on paths that have certain clearance, or minimal distance, from obstacles. Thus, the algorithm operates over problem instances which are robustly feasible as defined below.

**Defn. 2 (Robust Feasibility)** *An instance of the path planning problem ($C_{free}$, $q_{init}$, $q_{goal}$) is robustly feasible, if a $cl$-robust path exists that solves it, for some clearance $cl > 0$. A path $\pi \in \Pi$ is $cl$-robust, if $\pi$ lies entirely in the $cl$-interior of $C_{free}$.*

A naive way to define a graphical representation of $C_{\text{free}}$ is by considering the implicit, exhaustive graph $G(V, E)$, where all the elements of $C_{\text{free}}$ are nodes and all the collision free paths between them are edges. A compact data structure for answering shortest-path queries in continuous spaces must be able to identify which $C$-space points are not needed as roadmap nodes. Thus, a "roadmap spanner" is a subgraph $G_S(V_S \subset V, E_S \subset E)$ of this implicit, exhaustive graph of the continuous space and should satisfy the following properties:

1. All nodes in $G$ are connected with a path in $C_{\text{free}}$ to a node on $G_S$ (coverage).
2. $G_S$ has as many connected components as $G$ (connectivity).
3. All shortest paths on $G_S$ are no longer than $t$ times the corresponding shortest paths in $G$ (spanner property).

These properties allow for (a) arbitrary query points to connect to the roadmap, (b) paths to exist between any query points through $G_S$ that can be connected in $C_{\text{free}}$, and (c) asymptotic near-optimality for query points that lie on $G_S$. The objective is to also provide asymptotic near-optimality properties for query points that will not be lying on $G_S$. For such points, it is necessary to take into account the cost of connecting them to $G_S$, which gives rise to an additive term regarding the relative cost of the solution path computed by the spanner $G_S$ and the optimum solution cost on $G$. Overall, the objective is to guarantee the following property.

**Defn. 3 (Asympt. Near-Optimality with Additive Cost)** *An algorithm is asymptotically near-optimal with additive cost if, for a path planning problem ($C_{free}$, $q_{init}$, $q_{goal}$) and cost function $c : \Pi \to \mathbb{R}^{\geq 0}$ with a $cl$-robust optimal path of finite cost $c^*$, the probability it will find a path with cost $c \leq t \cdot c^* + \epsilon$, for a stretch factor $t \geq 1$ and additive error $\epsilon \geq 0$, converges to 1 as time approaches infinity.*

This paper presents algorithms that asymptotically converge to sparse data structures with the above property.

## 3.1 Sampling-based Planning Primitives

This work uses many of the same primitives that sampling-based motion planning typically employs. For instance, the proposed methods assume access to a uniform random sampler, which returns a point $\mathtt{q}$ at random in the $C$-space by following a uniform distribution given the topological constraints. Furthermore, it is assumed that the space is endowed with a distance function.

**Defn. 4 (Distance Function)** *The distance function $d$ takes two configurations in $C$-space and returns a real value $d(\mathtt{q}_i, \mathtt{q}_j) \to \mathbb{R}$ that expresses the distance of the two configurations in the absence of obstacles and satisfies metric properties.*

In order to construct the planning structure in $C_{\text{free}}$, sampling-based methods rely on being able to accept or reject samples based on whether a configuration brings the robot into collision with obstacles. In order to reject samples, the algorithms rely upon the availability of a collision checker. This work considers only configurations that are at least $cl$ away from obstacles.

**Defn. 5 (Validity Checker)** *Given an individual configuration $\mathtt{q}$, a validity checker returns whether $\mathtt{q}$ lies within the $cl$-interior of $C_{free}$.*

In order to create a roadmap, the sampled configurations must be connected by edges with the aid of a local planner.

**Defn. 6 (Local Planner)** *Given two configurations, a local planner returns a local path between the configurations in the absence of obstacles: $\mathtt{L}(\mathtt{q}_{begin}, \mathtt{q}_{end}) \to \pi_{\mathtt{L}}$, where $\pi_{\mathtt{L}}(0) = \mathtt{q}_{begin}$ and $\pi_{\mathtt{L}}(1) = \mathtt{q}_{end}$.*

In this work a straight line between $\mathtt{q}_{\text{begin}}$ and $\mathtt{q}_{\text{end}}$ in $C$-space is used. In order to add local paths as edges in the planning structure, it must be that for all $\mathtt{q} \in \mathtt{L}(\mathtt{q}, \mathtt{q}')$, $\mathtt{q} \in C_{\text{free}}$. There exist efficient and complete methods for checking if an entire path lies within $C_{\text{free}}$ (Schwarzer et al., 2005). Alternatively, a sampling-based process can be used as an approximation.

## 3.2 Sparse Roadmap Spanner Notation

To describe the proposed algorithms it will be helpful to introduce some new terminology. Every node in the sparse roadmap spanner $G_S$ will be selected so that it represents its local neighborhood (i.e., spanner paths that initiate in the local neighborhood will go through this node). The size of the neighborhood is limited by a visibility

radius $\Delta$, i.e., a new sample will attempt to connect to nodes only within such a distance. This visibility radius is introduced both for computational reasons (avoiding the collision checking of long paths) and so as to provide near-optimality guarantees (i.e., bounding the additive $\epsilon$ term that arises from the cost of connecting query points to the sparse roadmap spanner). Then for any configuration $\mathtt{q} \in C_{\text{free}}$ it is possible to compute its representative among the existing nodes in $G_S$ as follows:

**Defn. 7 (Representative)** *Given the nodes $V_S$ of the sparse roadmap spanner $G_S$, a configuration $\mathtt{q}$'s representative $\mathtt{v} \in V_G$ satisfies the following properties:*

- $d(\mathtt{q}, \mathtt{v}) \leq \Delta$ *for the visibility range* $\Delta$,
- $\mathtt{L}(\mathtt{q}, \mathtt{v}) \subset C_{free}$,
- $d(\mathtt{q}, \mathtt{v}) \leq d(\mathtt{q}, \mathtt{v}'), \forall\, \mathtt{v}' \in V_G$ *so that* $\mathtt{L}(\mathtt{q}, \mathtt{v}') \subset C_{free}$.

The representative of a configuration will be denoted as $\mathtt{rep}(\mathtt{q})$. This notion gives rise to the dual term of a visibility region.

**Defn. 8 (Visibility Region)** *The visibility region of a node $\mathtt{v} \in V_G$ is $\mathtt{vis}(\mathtt{v}) = \{\mathtt{q} \mid \mathtt{q} \in C_{free}, \mathtt{rep}(\mathtt{q}) = \mathtt{v}\}$.*

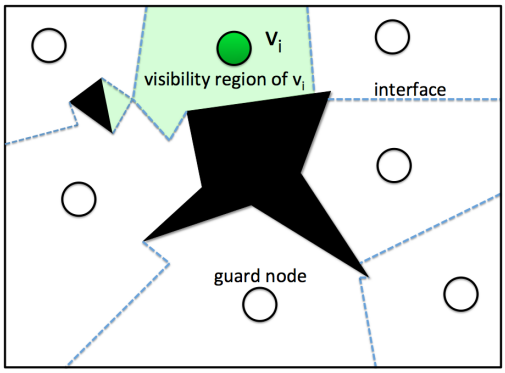


Figure 3: Visibility region of $\mathtt{v}_i$, i.e., the configurations connected to $\mathtt{v}_i$ which have it as their closest node. The figure ignores the effects of the visibility range $\Delta$.

The boundaries of a visibility region will arise from three separate conditions. The first is that the region is bounded by $\Delta$. Secondly, the region will be bounded by obstacle and visibility constraints, and lastly, it will be bounded by intersections with other visibility regions, called interfaces. An example of a visibility region can be seen in Fig. 3.

**Defn. 9 (Interface)** *Given the set of nodes $V_G$, an interface $\mathtt{i}(\mathtt{v}, \mathtt{v}')$, between two nodes $\mathtt{v}, \mathtt{v}' \in V_G$ is the shared boundary of their visibility regions:*

$$\mathtt{i}(\mathtt{v}, \mathtt{v}') = \mathtt{vis}(\mathtt{v}) \cap \mathtt{vis}(\mathtt{v}').$$

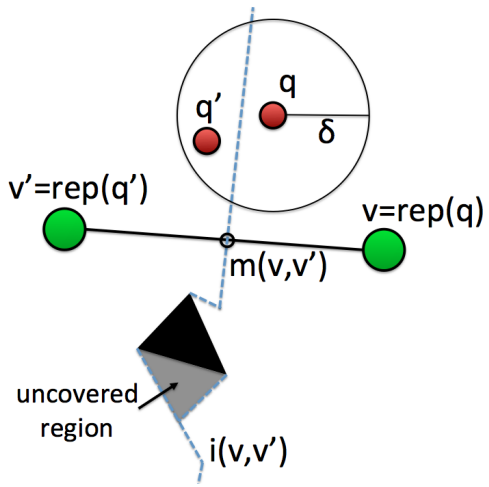


Figure 4: Two neighboring nodes $\mathtt{v}$ and $\mathtt{v}'$ define an interface $\mathtt{i}(\mathtt{v}, \mathtt{v}')$: the shared boundary of their visibility regions.

Figure 4 illustrates how an interface arises between two nodes. Given the definition of an interface it is possible to also describe the notion of support, which is also highlighted in the same figure.

**Defn. 10 (Support)** *Given the set of nodes $V_G$, a configuration $\mathtt{q}$ supports the interface $\mathtt{i}(\mathtt{v}, \mathtt{v}')$ if the following is true:*

- $\mathtt{rep}(\mathtt{q}) = \mathtt{v}$,
- $\exists\, \mathtt{q}' \in \mathcal{B}(\mathtt{q}, \delta) : \mathtt{rep}(\mathtt{q}') = \mathtt{v}' \wedge \mathtt{L}(q, q') \in C_{free}$.

In the above definition, the term $\mathcal{B}(\mathtt{q}, \delta)$ corresponds to the $\delta$-radius hyper-sphere centered at $\mathtt{q}$. It becomes apparent from the above definition that the set of configurations that support an interface corresponds to all these $C$-space points that are within distance $\delta$ from the interface. The notion of support is introduced because it is easier to discover configurations through a sampling process that support an interface than the the configurations along an interface, as the first set has positive measure, while the second one has zero measure. Finally, the notion of midpoint will be also useful to describe the properties of the proposed algorithms.

**Defn. 11 (Midpoint)** *The midpoint between two configurations $\mathtt{v}$ and $\mathtt{v}'$ along the local path $\mathtt{L}(\mathtt{v}, \mathtt{v}')$ will be denoted as $\mathtt{m}(\mathtt{v}, \mathtt{v}')$ and satisfies:*

$$\mathtt{m}(\mathtt{v}, \mathtt{v}') \in \mathtt{L}(\mathtt{v}, \mathtt{v}') \wedge d(\mathtt{v},\ \mathtt{m}(\mathtt{v}, \mathtt{v}')) = d(\mathtt{m}(\mathtt{v}, \mathtt{v}'),\ \mathtt{v}').$$

Note that if the local path $\mathtt{L}(\mathtt{v}, \mathtt{v}')$ is obstacle-free, then the midpoint $\mathtt{m}(\mathtt{v}, \mathtt{v}')$ lies on the interface $\mathtt{i}(\mathtt{v}, \mathtt{v}')$ between the two configurations, as shown in Figure 4.

# 4 Sparse Roadmap Spanners

Algorithm 1 provides a high-level description of the proposed framework for the generation of sparse roadmap spanners. The approach constructs a graph $G_S(V_S, E_S)$ by generating random valid samples in $C_{\text{free}}$, evaluating whether these configurations satisfy certain criteria to be added as nodes in $V_S$ and then connecting them with local paths. There are four methods for promoting a configuration to the sparse roadmap $G_S$ which are tested in order: "guards", "bridges", "interface nodes", and "shortcuts" as illustrated in Figure 2. A guard is added whenever a sample cannot be connected to any node already in $V_S$ with a collision-free path of length $\Delta$. A bridge is found whenever a configuration can be connected to multiple nodes in $V_S$ which are in disconnected components of $G_S$. Interface nodes are added when they reveal the existence of an interface between two spanner nodes, which do not share an edge in $E_S$. Shortcut nodes are added when a path is discovered in $C_{\text{free}}$ which is significantly shorter than corresponding paths through $G_S$. Two variations of the basic framework will be described later on: `SPARS` (from `SPA`rse `R`oadmap `S`panner algorithm) and `SPARS2`, which differ on how they implement the identification of shortcut nodes.

**Algorithm 1:** `Sparse_Roadmap_Spanner`$(M, t, k, \delta, \Delta)$

```
1  failures ← 0;
2  {G_S, G_D} ← Initialize_Graphs();
3  while failures < M do
4      q ← Sample_Configuration(δ, G_D);
5      W ← Visible_Guards(q, Δ, G_S);
6      if W == ∅ then
7          Add_Guard(q, G_S);
8      else if any two w ∈ W not connected then
9          Add_Connector(q, W, G_S);
10     else
11         Add_Interface_Node(q, Δ, G_S);
12     if q ∉ V(G_S) then
13         Add_Shortcut(q, t, k, δ, Δ, G_S, G_D);
14     if no change in G_S then
15         failures++;
16 return G_S;
```

The framework receives five input parameters:

- $M$: Used for the termination criterion of the approach, similar to the `Visibility-based PRM`.
- $t$: Corresponds to the stretch factor of the spanner.
- $k$: A parameter used by the `SPARS2` variation for sampling $k$ configurations in the local neighborhood $\mathcal{B}(\mathtt{q}, \delta)$.
- $\delta$ and $\Delta$: The two distance thresholds in the $C$-space described in the previous section. $\Delta$ corresponds to a "visibility" range for nodes in $V_S$, while $\delta$ is the radius of a local neighborhood for configurations in $C_{\text{free}}$ that defines the support of an interface. Typically $\delta << \Delta$.

The method first samples a valid configuration `q` by employing the uniform random sampler and the validity checker (line 4). Then the algorithm computes the set $W$ of existing nodes in $G_S$ that are within distance $\Delta$ and with which `q` can be connected (line 5), i.e.,

$$\forall\, w \in W : \mathtt{L}(\mathtt{q}, w) \in C_{\text{free}}.$$

Then, there are four possible reasons for which a newly sampled configuration can be added to $G_S$:

1. *Coverage:* The sample `q` is in a part of $C_{\text{free}}$ that is not covered by existing nodes in $V_S$, i.e., $W = \emptyset$ (lines 6-7). In this case, the sample `q` is added to the set of vertices: $V_S = V_S \cup \mathtt{q}$. The nodes added for $C$-space-space coverage will be called "guards". The purpose of these nodes is to ensure that whenever a query is given, the algorithm can connect the start and end query points to the sparse roadmap spanner with a collision-free local path.
2. *Connectivity:* The sample `q` is able to connect to at least two nodes that are otherwise disconnected: $\exists\, w_1, w_2 \in W$ so that there is no path in $G_S$ that connects $w_1$ and $w_2$ (lines 5,8-9). In this case, the sample `q` is added to the set of vertices: $V_S = V_S \cup \mathtt{q}$, and connected to the nodes:

$$\forall\, w \in W : E_S = E_S \cup \mathtt{L}(\mathtt{q}, w).$$

3. *Connecting Interfaces:* The sample `q` reveals the existence of an interface between two nodes that do not share an edge (lines 10-11). The analysis section will show that it is necessary for all pairs of nodes that share an interface to also be connected with an edge. The reasoning is that the algorithm compares paths between midpoints on the roadmap spanner and their relationship with optimum paths in $C_{\text{free}}$. If an interface exists between two nodes but they do not share an edge, then the midpoint is not on the roadmap spanner. Algorithm 2 details the steps that need to be taken in this case.

The algorithm `Add_Interface_Node` finds the two closest nodes $\mathtt{v}_1$ and $\mathtt{v}_2$ on the graph $G_S$ that are within $\Delta$ distance of the sample `q` ignoring obstacles(lines 1-3). Then if `q` can be connected to $\mathtt{v}_1$ and $\mathtt{v}_2$ but these two nodes are not directly connected (line 4), then the method has discovered the existence of an interface between two nodes, which do not share an edge. In this case, the method tries first to directly connect $\mathtt{v}_1$ and $\mathtt{v}_2$ (lines 5-6) and if this fails, then it adds `q` to $V_S$ and the local paths $\mathtt{L}(\mathtt{v}_1, \mathtt{q})$ and $\mathtt{L}(\mathtt{v}_2, \mathtt{q})$ to $E_S$ (lines 7-9).

**Algorithm 2:** `Add_Interface_Node`$(\mathtt{q}, \Delta, G_S)$

1 $N \leftarrow$ `Nearest_Guards`$(\mathtt{q}, \Delta, G_S)$;
2 $\mathtt{v}_1 \leftarrow \arg\min_{n \in N} d(\mathtt{q}, n)$;
3 $\mathtt{v}_2 \leftarrow \arg\min_{n \in N, n \neq v_1} d(\mathtt{q}, n)$;
4 **if** $\mathtt{L}(\mathtt{v}_1, \mathtt{q}), \mathtt{L}(\mathtt{q}, \mathtt{v}_2) \in C_{free} \wedge \mathtt{L}(\mathtt{v}_1, \mathtt{v}_2) \notin E_S$ **then**
5  **if** $\mathtt{L}(\mathtt{v}_1, \mathtt{v}_2) \in C_{free}$ **then**
6  $E_S \leftarrow E_S \cup \mathtt{L}(\mathtt{v}_1, \mathtt{v}_2)$;
7  **else**
8  $V_S \leftarrow V_S \cup \{\mathtt{q}\}$;
9  $E_S \leftarrow E_S \cup \{\mathtt{L}(\mathtt{v}_1, \mathtt{q}), \mathtt{L}(\mathtt{q}, \mathtt{v}_2)\}$;

Note that the method for discovering the existence of an interface does not directly correspond to the definition of the support of an interface as presented in the previous section and differs from the original presentation of the SPARS algorithm (Dobson et al., 2012). A method that would correspond to this definition would require to search the $\delta$-sized hyper-ball centered at q in order to identify whether there are configurations in $\mathcal{B}(\mathtt{q}, \delta)$ with different representatives than q. While this is a valid approach to detect an interface and will become necessary in the last part of the algorithm, it also more computationally expensive and takes longer time for the algorithm to discover interfaces. The method described here is a more efficient way to achieve the same objective that utilizes only information from a single $C$-space sample as well as the sample's connectivity properties with nodes of the sparse roadmap spanner. The analysis section will provide a proof that one sample is sufficient to reveal the existence of interfaces between nodes that do not share edges.

4. *Path Quality:* The fourth criterion is evaluated if none of the other ones has succeeded in augmenting the graph (line 12 of algorithm `Sparse_Roadmap_Spanner`). Its purpose is to evaluate whether the new sample q reveals that there is a shortest path in $C_{\text{free}}$ that is $t$ times shorter or more than the path on the sparse roadmap spanner which will be used to answer a similar query (line 13). This work describes two alternative ways to implement this criterion, one that utilizes a dense graph $G_D$ of configuration samples to estimate the shortest paths in $C_{\text{free}}$ (SPARS) and another one that avoids the memory requirement of storing $G_D$ and follows a conservative approximation for computing shortest paths in $C_{\text{free}}$ (SPARS2). The following sections will provide the details of each variation.

If a sample fails all criteria, then it is not added to $G_S$ and the parameter *failures* is incremented (lines 14-15). Should *failures* reach the threshold parameter $M$ (line 3), the algorithm terminates and returns the graph computed up to that point (line 16).

## 4.1 Using PRM* to Find Shortest $C_{\text{free}}$ Paths

The last criterion aims to guarantee that paths returned by the roadmap satisfy the spanner property relative to optimum paths in $C_{\text{free}}$. The challenge for function `Add_Shortcut` is to provide an algorithmic way for checking whether this is true. The idea in the proposed framework is that to achieve this by reasoning locally within the visibility region of each node $\mathtt{v} \in V_S$.

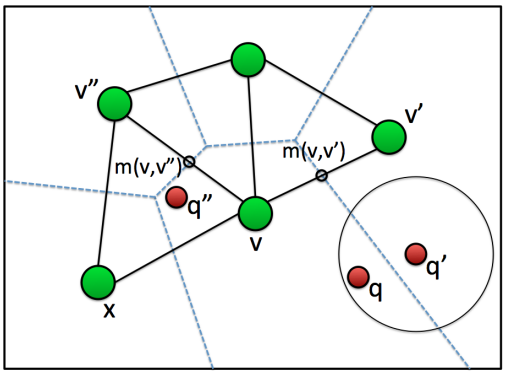


Figure 5: Configurations q and $\mathtt{q}'$ support $\mathtt{i}(\mathtt{v}, \mathtt{v}')$, while $\mathtt{q}''$ supports $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$. If the optimum path on the dense graph $\pi^*_D(q, q'')$ is $t$ times shorter than the length of the spanner paths $\pi_S(\mathtt{m}(\mathtt{v}', \mathtt{v}), \mathtt{m}(\mathtt{v}, \mathtt{v}''))$ or $\pi_S(\mathtt{m}(\mathtt{v}', \mathtt{v}), \mathtt{m}(\mathtt{v}, x))$, then configurations along $\pi^*_D(q, q'')$ are candidates for addition.

Consider the situation in Fig. 5 and a shortest path $\pi^*(\mathtt{q}_{begin}, \mathtt{q}_{end})$ in $C_{\text{free}}$ that goes through $\mathtt{vis}(\mathtt{v})$. Path $\pi^*$ can be partitioned into several segments, where each one is the intersection of $\pi^*$ with the visibility region of a node. The roadmap will satisfy the spanner property if it can provide a path $\pi_S(\mathtt{q}_{begin}, \mathtt{q}_{end})$ which is less than $t$ times the length of $\pi^*(\mathtt{q}_{begin}, \mathtt{q}_{end})$. Construct then the following spanner path to achieve this objective: replace each segment of the optimum path $\pi^*$ that goes through a visibility region $\mathtt{vis}(\mathtt{v})$, with the spanner path $\pi_S(\mathtt{m}(\mathtt{v}', \mathtt{v}), \mathtt{m}(\mathtt{v}, \mathtt{v}''))$ that connects the midpoints of the spanner edges connecting v with the nodes from whose visibility regions the optimum path enters and exits. For the first segment consider the path from $\mathtt{q}_{begin}$ to its representative $\mathtt{rep}(\mathtt{q}_{begin})$ and then to the midpoint of the edge connecting $\mathtt{rep}(\mathtt{q}_{begin})$ to the node that $\pi^*$ is crossing into its visibility region. Similarly for the last segment and $q_{end}$. It is then sufficient that each segment of the optimum path is no more than $t$ times shorter than the corresponding midpoint spanner path $\pi_S(\mathtt{m}(\mathtt{v}', \mathtt{v}), \mathtt{m}(\mathtt{v}, \mathtt{v}''))$. This is going to be certainly true, if the spanner path is shorter than $t$ times the shortest $C_{\text{free}}$ path between any two configurations on interfaces $\mathtt{i}(\mathtt{v}', \mathtt{v})$ and $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$.

The idea in SPARS is that it is possible to asymptotically compute all such shortest paths between interfaces by building in parallel with the roadmap a dense, asymptotically-optimal graph $G_D(V_D, E_D)$ using the PRM* algorithm. Every time that an interface path through the dense graph is revealed to be significantly shorter than the corresponding midpoint spanner path, the configurations along the interface path on the dense graph become candidates for addition to the roadmap. Thus, the algorithm operates as follows: Upon initialization of the roadmap $G_S$, it also initializes a dense graph $G_D$ (line 2 of routine Sparse_Roadmap_Spanner). Furthermore, every time that a configuration is sampled, it is immediately added as a node in $V_D$ and attempts are made to connect it to all nodes within a $\delta$ distance on the dense graph per the $\delta$-PRM* algorithm (Karaman and Frazzoli, 2011) (line 4 of routine Sparse_Roadmap_Spanner). Consequently, the structure $G_D$ will asymptotically converge to optimal $C_{\text{free}}$ paths. The Add_Shortcut function in SPARS uses the dense graph $G_D$ to compute interface paths for the representative of the current sampled configuration q.

**Algorithm 3:** Add_Shortcut$(\mathtt{q}, t, k, \delta, \Delta, G_S, G_D)$

```
1  v ← rep(q);
2  Q' ← Adjacent_Vertices(q, δ, G_D);
3  V' ← {v' | ∃ q' ∈ Q', so that v' = rep(q')};
4  for v' ∈ V' \ v do
5      for v'' ∈ V_S : ∃ i(v'', v) ∧
6      L(v'', v) ∈ E_G ∧ L(v'', v') ∉ E_G do
7          π_S ← Max_Spanner_Path(v, v', v'', G_S)
8          Q'' ← Interface_Support(v, v'');
9          π*_D ← argmin_{q''∈Q''} |π_D(q, q'')|;
10         if t · |π*_D| < |π_S| then
11             if L(v', v'') ∈ C_free then
12                 E_S ← E_S ∪ L(v', v'');
13             else
14                 Add_Path(G_S, {v' → π*_D → v''});
```

Algorithm 3 first computes the representative of q (line 1) and then finds the set $Q'$ of all neighbors of q in the dense graph within distance $\delta$ and their representatives $V'$ (lines 2-3). If there is a node $\mathtt{v}' \in V'$ that is different than the representative v of q, then q supports an interface per the definition of Section 3 (line 4). Then the algorithm considers all nodes $\mathtt{v}''$ in the roadmap spanner, which share an interface and an edge with v but do not share an edge with $\mathtt{v}'$ (lines 5-6). Note that it is possible for two nodes to share an edge but not an interface, as the algorithm ends up including edges that cross multiple visibility regions. The detection of an interface between two nodes v and $\mathtt{v}''$ in this algorithm utilizes information from the underlying dense graph. In particular, the algorithm detects if two configurations q and $\mathtt{q}''$, which share an edge in the dense graph $G_D$ have different representatives, i.e., $\mathtt{q} \in \mathtt{vis}(\mathtt{v})$ and $\mathtt{q}'' \in \mathtt{vis}(\mathtt{v}'')$, where $\mathtt{v} \neq \mathtt{v}''$. This means that there is an interface $i(\mathtt{v}, \mathtt{v}'')$. The situation is equivalent to what is displayed in Figure 5. If the two sparse nodes v and $\mathtt{v}''$ are not already connected with an edge, the third criterion will augment the roadmap spanner in order to bridge this interface. In this process, it is not necessary to consider vertices $\mathtt{v}''$ that share an edge with $\mathtt{v}'$. This is because the edge $\mathtt{L}(\mathtt{v}', \mathtt{v}'')$ acts as a shortcut to the midpoint spanner path $\pi_S(\mathtt{m}(\mathtt{v}', \mathtt{v}), \mathtt{m}(\mathtt{v}, \mathtt{v}''))$. In this case there is no reason to compare against the interface path from $\mathtt{i}(\mathtt{v}', \mathtt{v})$ to $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$.

**Algorithm 4:** Max_Spanner_Path$(\mathtt{v}, \mathtt{v}', \mathtt{v}'', G_S)$

```
1  Π_S ← { π_S( m(v', v), m(v, v'') ) };
2  for x : L(v, x), L(v'', x) ∈ E_S ∧ L(v', x) ∉ E_S do
3      if ∃i(x, v) then
4          Π_S ← Π_S ∪ { π_S( m(v', v), m(v, x) ) };
5  return argmax_{∀π∈Π_S} |π|;
```

Then for such a set of vertices $\mathtt{v}, \mathtt{v}', \mathtt{v}''$ (i.e., v connected to $\mathtt{v}'$ and $\mathtt{v}''$ but no edge between v and $\mathtt{v}''$) the spanner path $\pi_S$ is computed by calling function Max_Spanner_Path (line 6). Algorithm 4 provides the implementation for this procedure. Notice that the spanner path $\pi_S$ ends up being the maximum length path among $\pi_S(\mathtt{m}(\mathtt{v}', \mathtt{v}), \mathtt{m}(\mathtt{v}, \mathtt{v}''))$ and all paths of the form $\pi_S(\mathtt{m}(\mathtt{v}', \mathtt{v}), \mathtt{m}(\mathtt{v}, \mathtt{x}))$, where x are nodes that share an interface and an edge with v, share an edge with $\mathtt{v}''$ but do not share with $\mathtt{v}'$. In order to keep the description brief, the reason for considering these additional vertices x will become apparent during the analysis of the method.

Once the spanner path $\pi_S$ is found, function Add_Shortcut proceeds to find the corresponding shortest path between the interfaces $\mathtt{i}(\mathtt{v}', \mathtt{v})$ and $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$ given the addition of the new sample $q$ that has been shown to support $\mathtt{i}(\mathtt{v}', \mathtt{v})$. The algorithm first finds the set of configurations that supports the interface $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$ (line 8) and the shortest path $\pi^*_D$ on the dense graph between q and these configurations (line 9). Then the spanner property is checked between $\pi^*_D$ and $\pi_S$ (line 10). If it is violated, the algorithm first attempts to add a direct edge between nodes $\mathtt{v}'$ and $\mathtt{v}''$ (lines 11-12). If this is not possible, then the entire path $\{\mathtt{v}' \rightarrow \pi^*_D \rightarrow \mathtt{v}''\}$ is considered for addition (lines 13-14). The implementation of Add_Path adds two configurations along the input path that support the interfaces $\mathtt{i}(\mathtt{v}', \mathtt{v})$ and $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$ and then tries to smooth the remaining path as much as possible so as to minimize the

number of nodes and edges added to the sparse roadmap spanner. The addition of this path resolves the violation of the spanner property as argued in the analysis section.

## 4.2 Alternative to Storing a Dense Graph

Maintaining the dense graph $G_D$ is very costly in terms of memory requirements for the algorithm during the construction process of the sparse roadmap spanner. In order to reduce the computational footprint of SPARS, a variant is presented which removes the reliance on the dense graph. SPARS2 accomplishes this by employing conservative approximations of the shortest paths between interfaces $\mathtt{i}(\mathtt{v}', \mathtt{v})$ and $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$ and making use of some alternative bookkeeping information stored on the nodes of the planning structure. Note that a dense graph is no longer initialized by function Sparse_Roadmap_Spanner and when a sample q is generated in SPARS2 it is no longer added to a dense graph (lines 2 and 4 of Sparse_Roadmap_Spanner).

**Algorithm 5:** Add_Shortcut2($\mathtt{q}, t, k, \delta, \Delta, G_S$)

```
1  v ← rep(q);
2  (Q', V') ← (∅, ∅);
3  for k iterations do
4      q' ← Sample_Near(q, δ);
5      if L(q, q') ∈ C_free then
6          v' ← rep(q');
7          if ∄ v' then
8              Add_Guard(q', G_S);
9          else if v' ≠ v then
10             Q' ← Q' ∪ q',  V' ← V' ∪ v';
11 if V' ≠ ∅ then
12     for each v' ∈ V' and q' ∈ Q' do
13         Update_Points(q, q', v, v', G_S);
14         Update_Points(q', q, v', v, G_S);
15     Test_Add_Paths(v, G_S);
16     for each v' ∈ V' do
17         Test_Add_Paths(v', G_S);
```

The first use of the dense graph in SPARS was to detect samples that support an interface (line 2 of Add_Shortcut). Since the dense graph is no longer maintained, an alternative method is needed to detect samples that support interfaces. Lines 2 - 10 of Algorithm 5 provide a sampling-based method to achieve this objective. Each time a new configuration q is tested for the addition of a shortcut, $k$ additional samples $\mathtt{q}'$ are generated in its $\delta$-radius hyper-ball $\mathcal{B}(\mathtt{q}, \delta)$ (lines 3-4). If the local path $\mathtt{L}(\mathtt{q}, \mathtt{q}')$ is free and the representative of $\mathtt{q}'$ is different than that of q (lines 5-6,9-10), then the requirements for identifying two configurations that support an interface have been met. Note that in the case that $\mathtt{q}'$ cannot be connected to any existing node, a new "guard" has been discovered and needs to be added as such to the roadmap (lines 7-8).

Once it is detected that sample q supports an interface of its representative v, it is necessary to check whether it reveals the existence of a new interface path that is not covered by the corresponding spanner midpoint path. If the dense graph were available, it would be possible to iterate over all configurations that support another interface of v and compute shortest paths between them and q. While computing shortest paths in $C_{\text{free}}$ is not possible without the dense graph, it is still possible to detect samples that support interfaces as the previous paragraph indicated. Towards this objective, SPARS2 maintains for each node v and for each pair of neighbors $(\mathtt{v}', \mathtt{v}'')$ the following information (this information is stored on each node v of the corresponding graph data structure):

- A pair $P_{\mathtt{v}}(\mathtt{v}', \mathtt{v}'')$ of configurations that support the corresponding interfaces $\mathtt{i}(\mathtt{v}', \mathtt{v})$ and $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$, which both belong in $\mathtt{vis}(\mathtt{v})$, and define the shortest distance between any such pair of configurations.
- And a corresponding pair $\Xi_{\mathtt{v}}(\mathtt{v}', \mathtt{v}'')$ of configurations that support the interfaces $\mathtt{i}(\mathtt{v}', \mathtt{v})$ and $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$, where the first belongs in $\mathtt{vis}(\mathtt{v}')$ and the second belongs in $\mathtt{vis}(\mathtt{v}'')$, and are the samples that reveal that the configurations in $P_{\mathtt{v}}(\mathtt{v}', \mathtt{v}'')$ are supporting an interface.

**Algorithm 6:** Update_Points($\mathtt{q}, \mathtt{q}', \mathtt{v}, \mathtt{v}', G_S$)

```
1  for v'' ∈ V_S \ v, v' do
2      if ∃ i(v'', v) then
3          if L(v, v'') ∈ E_S and L(v', v'') ∉ E_S then
4              (ρ', ρ'') ← P_v(v', v'');
5              (ξ', ξ'') ← Ξ_v(v', v'');
6              if d(q, ρ'') < d(ρ', ρ'') then
7                  P_v(v', v'') = (q, ρ'');
8                  Ξ_v(v', v'') = (q', ξ'');
```

Algorithm 6 is responsible for updating this bookkeeping information, and is illustrated by Figure 6. Given a configuration q which belongs in $\mathtt{vis}(\mathtt{v})$ and a configuration $\mathtt{q}' \in \mathcal{B}(\mathtt{q}, \delta)$ which belongs in $\mathtt{vis}(\mathtt{v}')$ (both configurations support $i(\mathtt{v}, \mathtt{v}')$)) the algorithm considers all vertices $\mathtt{v}''$, which share an interface and an edge with v but not an edge $\mathtt{v}'$ (lines 1-3). Then the algorithm retrieves the previously stored pair of configurations $(\rho', \rho'')$ that support $\mathtt{i}(\mathtt{v}', \mathtt{v})$ and $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$ from the side of v, which correspond to the shortest distance among all such configurations (line 4). The corresponding $\Xi_{\mathtt{v}}(\mathtt{v}', \mathtt{v}'')$ pair

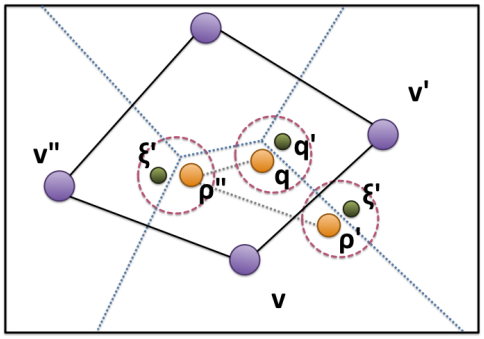


Figure 6: An example of the relative locations of the configurations used in the Algorithm 6. In this example, the new sample q is closer to $\rho''$ than $\rho'$ was, so the pair of configurations for the connection between $\mathtt{v}'$ and $\mathtt{v}''$ via v will be updated.

is also retrieved (line 5). If the new sample q is closer to the best representative of the interface $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$ (line 6), then the bookkeeping information is being updated with q and $\mathtt{q}'$. Note that function Update_Points is called to update not only the information of v (line 13 of Add_Shortcut2) but also for the neighboring node $\mathtt{v}'$ of whose interface configuration q supports (line 14). A notable concern is how large this extra bookkeeping information becomes. It is expected that as the dimensionality of a problem increases, the space requirements in order to maintain this information will also increase significantly. It seems, however, that for problems up to 6 dimensions, the memory requirements are quite manageable, as detailed in Section 6.

Once the bookkeeping information is updated, algorithm Add_Shortcut2 proceeds to check if the new sample q revealed two interfaces that are closer than the corresponding spanner midpoint path by calling function Test_Add_Paths (lines 15-17). Note that this function is called only if Update_Points had to update the bookkeeping information.

Algorithm 7 provides the implementation of Test_Add_Paths and in many ways it is similar to the second part of Add_Shortcut from SPARS. It reasons again for all vertices $\mathtt{v}''$, which share an interface and an edge with v but not an edge $\mathtt{v}'$ (lines 1-3). For each such vertex $\mathtt{v}''$ it retrieves from the bookkeeping information the shortest distance between two configurations in $\mathtt{vis}(\mathtt{v})$ that support $\mathtt{i}(\mathtt{v}', \mathtt{v})$ and $\mathtt{i}(\mathtt{v}, \mathtt{v}'')$ (lines 4-6). This distance is a conservative approximation of the shortest path in $C_{\text{free}}$ between these two interfaces, i.e.,

**Algorithm 7:** Test_Add_Paths(v, $G_S$)

```
1  for v' ∈ V_S : L(v, v'') ∈ E_S do
2      for v'' ∈ V_S : ∃ i(v'', v) ∧
3      L(v, v'') ∈ E_S ∧ L(v', v'') ∉ E_S do
4          (ρ', ρ'') ← P_v(v', v'');
5          (ξ', ξ'') ← Ξ_v(v', v'');
6          |π̂*_D| ← d(ρ', ρ'');
7          π_S ← Max_Spanner_Path(v, v', v'', G_S);
8          if t * |π̂*_D| < |π_S| then
9              if L(v', v'') ∈ C_free then
10                 E_S ← E_S ∪ L(v', v'');
11             else
12                 Add_Path(G_S, {v' → ξ' → ρ' →
                   v → ρ'' → ξ'' → v''});
```

it will always converge over time to something shorter than the true shortest path. The corresponding spanner midpoint path $\pi_S$ is computed as in SPARS (line 7) and then the spanner property is evaluated (line 8). If violated, an attempt to directly connect $\mathtt{v}'$ and $\mathtt{v}''$ is made (lines 9-10). If this is not successful, then the entire path $\{\mathtt{v}' \rightarrow \xi' \rightarrow \rho' \rightarrow \mathtt{v} \rightarrow \rho'' \rightarrow \xi'' \rightarrow \mathtt{v}''\}$ is considered for addition (lines 11-12). The implementation of Add_Path adds $\xi'$ and $\xi''$ and then tries to smooth the remaining path as much as possible so as to minimize the number of nodes and edges added to the sparse roadmap spanner.

# 5 Analysis

The discussion on the properties initially relates to the entire framework for the generation of sparse roadmap spanners and then focuses on the SPARS (Section 4.1) and SPARS2 (Section 4.2) variations. A series of lemmas argues the probabilistic completeness of the method, that paths returned by the proposed approach converge to near-optimal and that the probability of adding nodes to the structure goes to zero as time progresses.

## 5.1 Probabilistic Completeness

The approach is equivalent to $\mathtt{Visibility-based\ PRM}$ (Simeon et al., 2000) with regards to coverage and connectivity. From this equivalence, it is possible to argue that the framework achieves probabilistic completeness.

**Thm. 1 (Coverage)** *For all* $\mathtt{q} \in C_{free} : \exists\ \mathtt{v} \in V_S$ *so that* $\mathtt{L}(\mathtt{q}, \mathtt{v}) \in C_{free}$ *with probability approaching* 1 *as* $M$ *goes to infinity in the* Sparse_Roadmap_Spanner *algorithm.*

The argument to support the above statement can be found in the presentation of the Visibility − based PRM (Simeon et al., 2000) as the framework adds all the nodes that would be added by this method and follows a similar termination condition. At a high level, each new guard inserted in $G_S$ increases the coverage of $C_{\text{free}}$ and the probability of generating configurations in non-covered regions decreases over time. The algorithm is then guaranteed to terminate for any finite input value $M$. When it stops, a probabilistic estimation of the percentage of free space not covered by spanner nodes is $\frac{1}{M}$, given uniform sampling. This means that future attempts to add spanner nodes will succeed with probability $(1 - \frac{1}{M})$. Consequently, as $M$ goes to infinity, the resulting graph covers the entire space.

Note that relative to the Visibility − based PRM, the probability $(1 - \frac{1}{M})$ is more conservative, as the methods for generating sparse roadmap spanners assume a visibility range limit $\Delta$ for graph nodes. The work on Visibility − based PRM has shown that even for fairly complicated problems in $SE(3)$, a relatively small number of guards is needed to probabilistically cover the space. Connectivity properties of the resulting sparse roadmap spanner can be argued in a similar manner.

**Thm. 2 (Connectivity)** *For all* $\mathtt{v}, \mathtt{v}' \in V_S$ *that are connected with a collision-free path in* $C_{free}$, $\exists\ \pi_S(\mathtt{v}, \mathtt{v}')$ *which connects them on* $G_S$ *with probability* 1 *as* $M$ *goes to infinity.*

The framework explicitly handles checking for connectivity in lines 8-9 of Algorithm 1. The algorithm adds an edge or a node every time it detects there is a way to connect two disconnected components. The probability of sampling a configuration that will connect such disconnected components of the graph depends on the environment. Narrow passages will make this connection more challenging. The probability of connecting any two disconnected components is 1, as the value of $M$ goes to infinity, if the following assumption is true:

**Assumption 1** *For any* $\mathtt{v}, \mathtt{v}' \in C_{free}$, *if the set* $Q$ *of configurations* $\mathtt{q} \in Q$ *for which the following properties hold is non-empty:*

- $\mathtt{L}(\mathtt{q}, \mathtt{v}) \in C_{free}$, $d(\mathtt{q}, \mathtt{v}) < \Delta$,
- $\mathtt{L}(\mathtt{q}, \mathtt{v}') \in C_{free}$, $d(\mathtt{q}, \mathtt{v}') < \Delta$,

*then* $Q$ *has non-zero measure.*

It can be argued that the above assumption relates to the definition of an expansive space (Hsu et al., 1999). The combination of the last two theorems provides probabilistic completeness, at least when the algorithm samples configurations q in a uniform way.

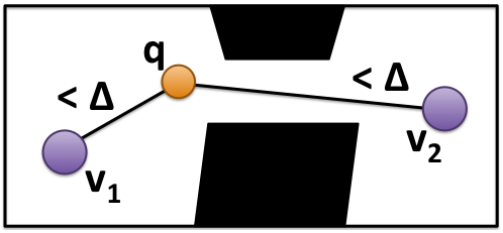


Figure 7: One sample can reveal interfaces.

In order to prove certain properties of the algorithm, it must be that all vertices in the planning structure which share an interface must also share an edge on the roadmap. The following theorem argues that this is indeed the case.

**Thm. 3 (Connected Interfaces)** *For all* $\mathtt{v}_1, \mathtt{v}_2 \in V_S$ *which share an interface, then* $\mathtt{L}(\mathtt{v}_1, \mathtt{v}_2) \in E_S$ *with probability approaching* 1 *as* $M$ *goes to infinity.*

**Proof:** In order for the algorithm to detect the existence of an interface between two nodes it has to be that a sample q can connect to its two closest guards $\mathtt{v}_1$ and $\mathtt{v}_2$ as in Figure 7 (lines 1-4 of Add_Interface_Node). If this is the case, then there must be an interface between $\mathtt{v}_1$ and $\mathtt{v}_2$. Consider a moving configuration $\mathtt{q}^*$ along the local path from q to $\mathtt{v}_2$. Moving along this path guarantees that visibility with $\mathtt{v}_2$ is maintained (i.e., $\mathtt{L}(\mathtt{q}^*, \mathtt{v}_2) \in C_{\text{free}}$). There are two cases:

(a) either visibility with $\mathtt{v}_1$ is maintained until $\mathtt{q}^*$ becomes equidistant with $\mathtt{v}_1$ and $\mathtt{v}_2$ or

(b) at some point visibility with $\mathtt{v}_1$ is lost before $\mathtt{q}^*$ becomes equidistant.

In the first case, an interface exists at the point $\mathtt{q}^*$ becomes equidistant with $\mathtt{v}_1$ and $\mathtt{v}_2$. It cannot be than any other node $\mathtt{v}_3$ will be closer to $\mathtt{q}^*$ than $\mathtt{v}_1$ and $\mathtt{v}_2$. If there were, then $\mathtt{v}_3$ would have been closer to q than $\mathtt{v}_2$, which is not true. In the second case, the interface exists at the point where the visibility with $\mathtt{v}_1$ is lost. For the same reasons, no other node $\mathtt{v}_3$ will be closer at this point than $\mathtt{v}_1$. In order to be able to sample point q in the first place, it is sufficient that Assumption 1 holds. The above discussion implies that the method is able through a sampling process to detect all pairs of spanner nodes that share an edge but not an interface. The algorithm will try to add an edge between nodes $\mathtt{v}_1$ and $\mathtt{v}_2$. If this fails, then q will be added and will be connected to $\mathtt{v}_1$ and $\mathtt{v}_2$. The addition of q might introduce new interfaces that are not intersected by edges. It will not be the case, however, that these newly created interfaces will always have an obstacle preventing a direct connection between the two vertices which impose it. At some point if vertices are added for this reason,

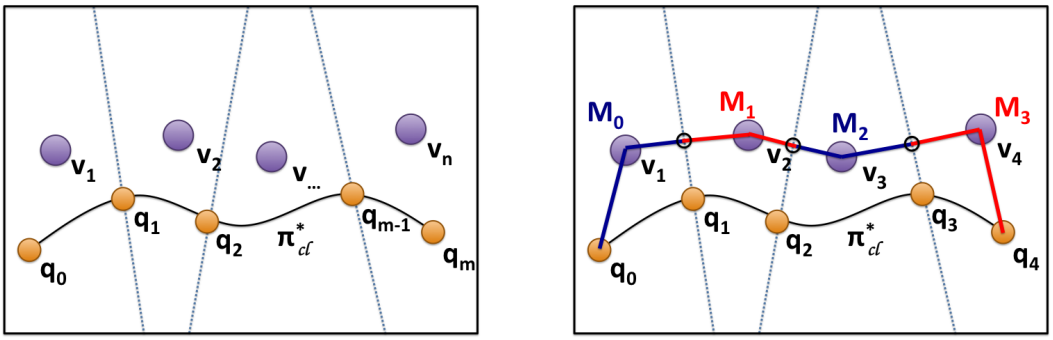


Figure 8: (left) All configurations along the optimal path $\pi^*_{cl}(\mathtt{q}_0, \mathtt{q}_m)$ will be eventually covered by a node in $G_S$. (right) A path between all spanner nodes covering $\pi^*_{cl}(\mathtt{q}_0, \mathtt{q}_m)$ will also be created. The figure shows the decomposition of path $\pi_S(\mathtt{q}_0, \mathtt{q}_m)$ into "midpoint paths" $M_{i-1}$ that exist only in a single visibility region. Each $M_{i-1}$ covers the path $\pi^*_{cl}(\mathtt{q}_i, \mathtt{q}_{i+1})$.

the spanner nodes will be closer than $cl$ and connections between them are guaranteed to be collision-free. In practice, most roadmap nodes will be connected before getting as close as $cl$. □

## 5.2 Path Quality

The following discussion focuses on showing asymptotic near-optimality properties.

**Lemma 1 (Coverage of Optimal Paths by $G_S$)** *Consider an optimal path $\pi^*_{cl}(\mathtt{q}_0, \mathtt{q}_m)$ in $C_{free}$. The probability of having a sequence of nodes in $S$, $V_\pi = (\mathtt{v}_1, \mathtt{v}_2, ..., \mathtt{v}_n)$ with the following properties approaches 1 as $M$ goes to infinity:*

- $\forall \mathtt{q} \in \pi^*_{cl}(\mathtt{q}_0, \mathtt{q}_m),\ \exists\ \mathtt{v} \in V_\pi :\ \mathtt{L}(\mathtt{q}, \mathtt{v}) \in C_{free}$
- $\mathtt{L}(\mathtt{q}_0, \mathtt{v}_1) \in C_{free}$ *and* $\mathtt{L}(\mathtt{q}_m, \mathtt{v}_n) \in C_{free}$
- $\forall\ \mathtt{v}_i, \mathtt{v}_{i+1} \in V_\pi, \mathtt{L}(\mathtt{v}_i, \mathtt{v}_{i+1}) \in E$.

The above lemma is a direct outcome of Theorems 1 and 3 regarding coverage and connected interfaces. Consider now a decomposition of the path $\pi_S(\mathtt{q}_0, \mathtt{q}_m)$ on $G$ through $V_\pi$ into sub-paths $\{M_0, M_1, \ldots, M_{m-1}\}$ as shown in Figure 8, where $M_i$ is the path between the midpoint $\mathtt{m}(\mathtt{v}_i, \mathtt{v}_{i+1})$ and $\mathtt{m}(\mathtt{v}_{i+1}, \mathtt{v}_{i+2})$, $M_0$ connects $\mathtt{q}_0$ to $\mathtt{v}_1$ and then $m(v_1, v_2)$ and $M_{m-1}$ is the corresponding last segment. Then the following can be shown:

**Lemma 2 (Additive Connection Cost)** *The sum of the lengths of segments $M_0$ and $M_{m-1}$ for a path from $\mathtt{q}_0$ to $\mathtt{q}_m$ through $G_S$ is upper bounded by $4 \cdot \Delta$.*

**Proof:** The cost for connecting samples to the spanner is at most $\Delta$ for both the start and final positions, as we know that $\mathtt{q}_0$ and $\mathtt{q}_m$ lie in the visibility region of $\mathtt{v}_1$ and $\mathtt{v}_m$ respectively. Note that the remainder of segment $M_0$ that connects $\mathtt{v}_1$ to $\mathtt{v}_2$ is at most $\Delta$ as two spanner nodes that share an interface cannot be further away than $2 \cdot \Delta$ and $M_0$ terminates at the midpoint between $\mathtt{v}_1$ and $\mathtt{v}_2$. The same is true for $M_{m-1}$, which implies that each of one of this two segments cannot be longer than $2 \cdot \Delta$ and the sum of their lengths is upper bounded by $4 \cdot \Delta$.□

An important property of the midpoint segments, $M_i$ is that they satisfy the spanner property for $C_{\text{free}}$ paths that go through the corresponding visibility region. Reasoning about the spanner property at a local level allows for the properties to be shown globally.

**Lemma 3 (Spanner Property of $G_S$ over $C_{\text{free}}$)** *All segments $M_i$ ($i \in [1, m-2]$) have length bounded by $t \cdot |\pi^*_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)|$, where $\mathtt{q}_i$ lies at the intersection of $\pi^*_{cl}(\mathtt{q}_0, \mathtt{q}_m)$ with interface $\mathtt{i}(\mathtt{v}_i, \mathtt{v}_{i+1})$.*

In order to prove this lemma, it must now be shown separately for SPARS and SPARS2 as they take different approaches to adding nodes for the sake of path quality. A proof is provided first for the SPARS method.

**Proof for SPARS:** The following discussion relates to Figures 9 and 10. Given Lemma 1, the edges $\mathtt{L}(\mathtt{v}_{i-1}, \mathtt{v}_i)$ and $\mathtt{L}(\mathtt{v}_i, \mathtt{v}_{i+1})$ are in $E_S$, at least as $M$ goes to infinity. Assuming $\pi^*_{cl}$ travels through the region of node $\mathtt{v}_i \in V_\pi$, there are three possible cases:

(a) $\mathtt{L}(\mathtt{v}_{i-1}, \mathtt{v}_{i+1}) \notin E_S$ (Figure 9(left)),
(b) $\mathtt{L}(\mathtt{v}_{i-1}, \mathtt{v}_{i+1}) \in E_S$, where $\mathtt{L}(\mathtt{v}_{i-2}, \mathtt{v}_i) \notin E_S$ and $\mathtt{L}(\mathtt{v}_i, \mathtt{v}_{i+2}) \notin E_S$ (Figure 9(right)), and
(c) $\mathtt{L}(\mathtt{v}_{i-1}, \mathtt{v}_{i+1}) \in E_S$ and $\mathtt{L}(\mathtt{v}_{i-2}, \mathtt{v}_i) \in E_S$ or $\mathtt{L}(\mathtt{v}_i, \mathtt{v}_{i+2}) \in E_S$ (Figure 10).

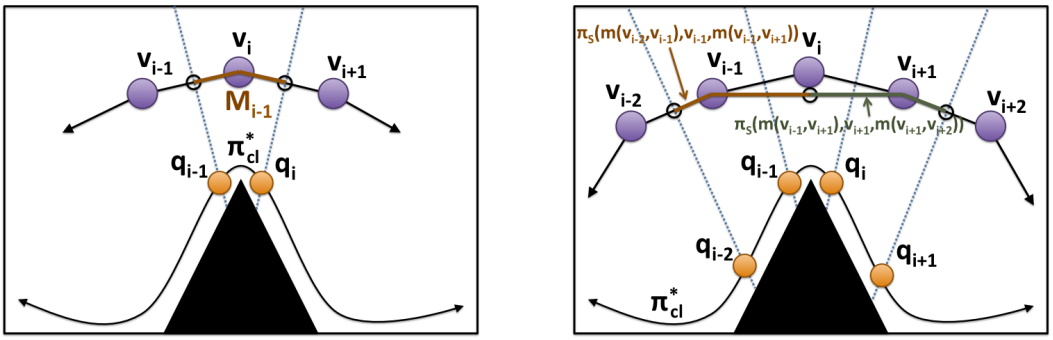


Figure 9: (left) The path from $i(\mathtt{v}_{i-1}, \mathtt{v}_i)$ to $i(\mathtt{v}_i, \mathtt{v}_{i+1})$ via $S$ must satisfy the spanner property for path $\pi^*_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)$. (right) If there is a path between $\mathtt{v}_i$'s neighbors, then the paths $\pi(m(\mathtt{v}_{i-2}, \mathtt{v}_{i-1}), \mathtt{v}_{i-1}, m(\mathtt{v}_{i-1}, \mathtt{v}_{i+1}))$ and $\pi(m(\mathtt{v}_{i+2}, \mathtt{v}_{i+1}), \mathtt{v}_{i+1}, m(\mathtt{v}_{i+1}, \mathtt{v}_{i-1}))$ must be checked against $\pi^*_{cl}(\mathtt{q}_{i-2}, \mathtt{q}_{i-1})$ and $\pi^*_{cl}(\mathtt{q}_i, \mathtt{q}_{i+1})$.

In the first case, the algorithm needs to check $|M_i|$ against $|\pi^*_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)|$. In the case of SPARS, the optimal path $\pi^*_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)$ is asymptotically approximated by the dense graph, so the shortest path through $D$ is used for the true optimum, and is denoted as $\pi^D_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)$. If it were found that $|M_i| > t \cdot \pi^D_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)$, the method would have passed the check on line 10 of Algorithm 3, and a shortcut path would have been added to the graph: either it would have added $\mathtt{L}(\mathtt{v}_{i-1}, \mathtt{v}_{i+1})$ in which scenario the second or third cases of this proof applies, or a dense path which includes samples from $D$ would have been included in $G_S$. This means that the representative sequence $V_\pi$ would have been different for this optimal path.

The second case requires a more careful examination. In particular, the algorithm compares $\pi^D_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)$ not only with the segment $M_i$ but also with all spanner paths from $\mathtt{m}(\mathtt{v}_{i-1}, \mathtt{v}_i)$ to the interfaces of all neighbors of $\mathtt{v}_i$, which are not connected to $\mathtt{v}_{i-1}$ but share an interface with $\mathtt{v}_{i+1}$. In the context of Figure 9(right) this allows to check whether path $\pi(\mathtt{m}(\mathtt{v}_{i-2}, \mathtt{v}_{i-1}), \mathtt{v}_{i-1}, \mathtt{m}(\mathtt{v}_{i-1}, \mathtt{v}_{i+1}))$ satisfies the spanner property for path $\pi^D_{cl}(\mathtt{q}_{i-2}, \mathtt{q}_{i-1})$ and that path $\pi(\mathtt{m}(\mathtt{v}_{i+2}, \mathtt{v}_{i+1}), \mathtt{v}_{i+1}, \mathtt{m}(\mathtt{v}_{i+1}, \mathtt{v}_{i-1}))$ satisfies the spanner property for $\pi^D_{cl}(\mathtt{q}_i, \mathtt{q}_{i+1})$. If they do, then it does not matter if segment $M_{i-1}$ satisfies the spanner property for path $\pi^D_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)$, because $\pi(\mathtt{m}(\mathtt{v}_{i-2}, \mathtt{v}_{i-1}), \mathtt{v}_{i-1}, \mathtt{m}(\mathtt{v}_{i-1}, \mathtt{v}_{i+1}))$ and $\pi(\mathtt{m}(\mathtt{v}_{i+2}, \mathtt{v}_{i+1}), \mathtt{v}_{i+1}, \mathtt{m}(\mathtt{v}_{i+1}, \mathtt{v}_{i-1}))$ cover all three consecutive subpaths of the optimum path. Otherwise, the spanner must have been expanded.

The last case is illustrated by Figure 10, which has two subcases: the solution returned by SPARS traveling from $\mathtt{v}_{i-2}$ to $\mathtt{v}_{i+2}$ does so through path $\pi_{a,b,c}$ or through path $\pi_{d,e,f}$. In the case that the path returned is $\pi_{a,b,c}$, the situation is case 2 of the proof and there is no concern. If however, the path returned is $\pi_{d,e,f}$, then it is necessary to show that $|\pi_e| \leq$
$t \cdot (|\pi^D_{cl}(\mathtt{q}_{i-2}, \mathtt{q}_{i-1})| + |\pi^D_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)| + |\pi^D_{cl}(\mathtt{q}_i, \mathtt{q}_{i+1})|)$.

It is known that $|\pi_b| \leq t \cdot |\pi^D_{cl}(\mathtt{q}_{i-2}, \mathtt{q}_{i-1})| + |\pi^D_{cl}(\mathtt{q}_i, \mathtt{q}_{i+1})|$ from case 2. It is also known that $|\pi_{d,e,f}| \leq |\pi_{a,b,c}|$ or it would otherwise not have been returned by SPARS. Furthermore, the construction of these segments enforces that $|\pi_a| \leq |\pi_d|$ and $|\pi_c| \leq |\pi_f|$, as the endpoints are the intersections with $\mathtt{i}(\mathtt{v}_{i-2}, \mathtt{v}_{i-1})$ for $\pi_a$ and $\pi_d$, and $\mathtt{i}(\mathtt{v}_{i+1}, \mathtt{v}_{i+2})$ for $\pi_c$ and $\pi_f$. By combining and substituting these inequalities, the following expression arises:

$$|\pi_e| \leq |\pi_a| + |\pi_b| + |\pi_c| - |\pi_d| - |\pi_f|.$$

Simplifying this result yields:

$$|\pi_e| \leq t \cdot (|\pi^D_{cl}(\mathtt{q}_{i-2}, \mathtt{q}_{i-1})| + |\pi^D_{cl}(\mathtt{q}_i, \mathtt{q}_{i+1})|) \leq$$
$$t \cdot (|\pi^D_{cl}(\mathtt{q}_{i-2}, \mathtt{q}_{i-1})| + |\pi^D_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)| + |\pi^D_{cl}(\mathtt{q}_i, \mathtt{q}_{i+1})|). \quad (2)$$

Consequently, this last case is not an issue for the relationship between spanner paths and $C_{\text{free}}$ paths. □

**Proof for SPARS2:** There are two points which must be addressed for SPARS2, as it no longer has dense graph information. SPARS2 must (a) use a conservative approximation for $|\pi^*_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)|$, and (b) introduce configurations to approximate dense paths, which would have otherwise been returned by the dense graph.

First, the approximation used for $\pi^*_{cl}(\mathtt{q}_{i-1}, \mathtt{q}_i)$ must be an underestimation of the true cost. SPARS2 employs $d(\mathtt{q}_{i-1}, \mathtt{q}_i)$ as the approximation of this cost. It holds that

this will always be an underestimation of the true optimal path as given the absence of obstacles, the true optimal paths are the local paths $\mathtt{L}(\mathtt{q}_{i-1}, \mathtt{q}_i)$. The presence of obstacles will serve only to detour this path and make it longer. Whenever SPARS2 is checking a midpoint path $M_i$ for the spanner property then, it will report that the paths are violating the spanner property more often then they may actually be.

The construction of the approximate optimal paths may be considerably worse than the optimal path which travels through the regions represented by the midpoint paths. In this case, even with powerful smoothing techniques, the resulting path may still not satisfy the spanner property given the conservative approximation of the true optimal path. The argument is that this is not problematic, as the addition of this approximate dense path will create new visibility regions through which new candidate midpoint paths can be evaluated in future iterations of the algorithm. Eventually, the direct connection between two interfaces will be collision free, and in this case, the conservative approximation of the optimal path length is the true optimal path length. Furthermore, because this path is collision-free, the smoothing operation can simply add $\mathtt{L}(\mathtt{q}_{i-1}, \mathtt{q}_i)$ as a dense path.

From this reasoning, cases 1 and 2 from the previous proof still hold for SPARS2. Case 3 also holds for both algorithms without loss of generality, as it is a property inherent to the planning structure and does not reason about optimal path information. □

Given the above sequence of lemmas, it is then possible to combine them and argue the following theorem:

**Thm. 4 (Asymptotic Near-Optimality w/Additive Cost)** *As $M$ goes to infinity: $\forall\ \mathtt{q}_0, \mathtt{q}_m \in C_{free} : |\pi_S(\mathtt{q}_0, \mathtt{q}_m)| < t \cdot |\pi^*_{cl}(\mathtt{q}_0, \mathtt{q}_m)| + 4 \cdot \Delta$.*

## 5.3 Rate of Node Addition

An important concern with sparse roadmap spanners is whether they will be adding a significant number of nodes in the final data structure. An additional assumption relates to the sampling process used by the algorithm, which is necessary for showing that paths will not be infinitely added to $G_S$.

**Assumption 2** *No sample $\mathtt{q}$ is within distance $cl$ from obstacles. No roadmap node $\mathtt{v} \in V_S$ is within $cl$-distance from obstacles.*

The following theorem argues that the rate of node addition decreases over time, which allows for the reasonable termination criterion the proposed framework employs in line 3 of Algorithm 1.

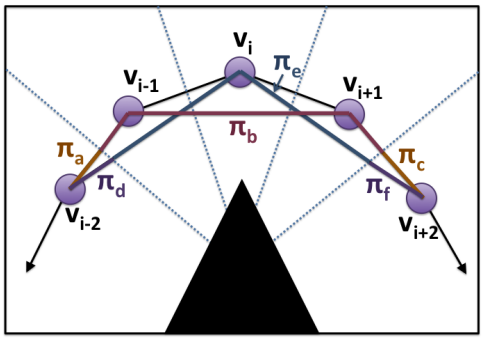


Figure 10: In the case that $\mathtt{v}_i$ depends on its neighbors for checking the spanner property which rely on $\mathtt{v}_i$, it must be that $|\pi_e| < |\pi_b|$.

**Thm. 5 (Rate of Node Addition)** *As the number of iterations increases in Algorithm 1, the probability of adding a node to $G_S$ goes to* 0.

**Proof Sketch** This proof sketch will rely on Assumption 1, as in certain poorly-behaved spaces, such as fractal spaces, it is impossible to even guarantee coverage with a finite set of samples. There are four possible reasons for promoting nodes to $G_S$; thus, it must be shown that all four of these criteria will eventually have no reason to add nodes to the planning structure.

**Nodes for coverage** are added when a sample $\mathtt{q}$ lies outside the visibility range of all existing nodes in $V$. Theorem 1 already argues that the probability of adding guards diminishes to zero as the number of iterations increases.

**Nodes for connectivity** are added when a sample $\mathtt{q}$ connects two disconnected components of the planning structure. Eventually, enough nodes for ensuring coverage will be added in $G$, and thus, these nodes must be connected. As it is not possible for there to be an infinite number of connected components of $C_{\text{free}}$, the number of samples needed to connect such disconnected components is finite and will be eventually added. to $\mathtt{v}'$.

**Nodes for ensuring interfaces have edges** are added when a sample reveals two nodes in the planning structure share an interface but not an edge. It has already been argued in Theorem 3 that the algorithm will connect all interfaces, thus nodes will stop being added at some point for this purpose. The reader might argue that the addition of nodes to reduce the number of interfaces, which do not share an edge, will actually generate new interfaces that do not have edges. This is in fact the case; however, it will not be the case that for all of these newly generated inter-

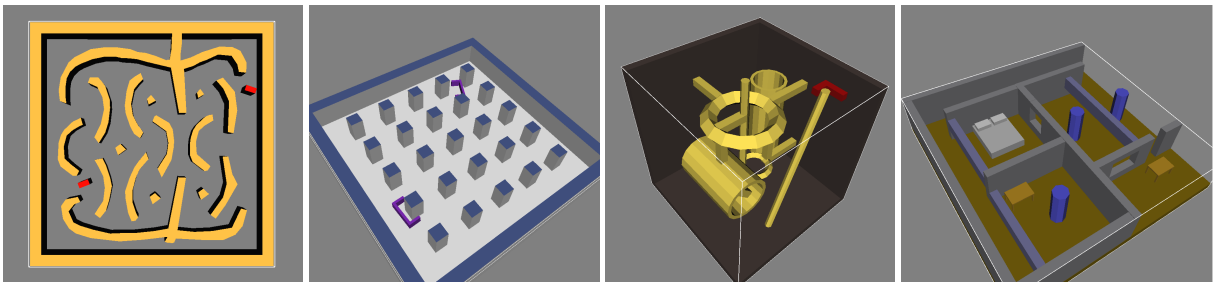

Figure 11: The four environments used for benchmarking the algorithms. From left to right are the "Maze", "Pegs", "Abstract" and "Beam Site" environments. Planning was performed in both $SE(2)$ ("Maze","Pegs") and $SE(3)$ ("Abstract","Beam Site") on rigid body systems.

faces there will be an obstacle preventing a direct connection from v

**Nodes for ensuring path quality** are added when a sample q supports an interface and reveals a path through $C_{\text{free}}$ that is significantly shorter than the corresponding path in the planning structure. In the case of SPARS, these paths correspond to dense paths through $D$, while in SPARS2, they are reconstructed paths. When adding such a path to $G$, the framework will first try to smooth this path. In some cases, the path cannot be smoothed, which will result in the addition of nodes to $G$. It is assumed, however, that samples are drawn from the $cl$-interior of $C_{\text{free}}$ given Assumption 2. This prevents paths from getting infinitely close to obstacles, and allows direct connections to be made between vertices once they get close enough. Therefore, both methods will eventually stop adding nodes in this fashion. □

### 5.4 Space Requirements

It is important to reason about the space requirements of the proposed SPARS algorithms upon construction. While the objective is to return a sparse data structure, this should not take place at the expense of significantly increasing the space requirements of the method during the construction process because of the book-keeping information utilized by the approaches.

The next section will show that experimentally the algorithms seem to have relatively low memory requirements, at least up to six dimensions. It is well understood that as the dimensionality of a problem increases, the number of potential neighbors for a node increases exponentially. This would indicate that the bookkeeping information stored on the nodes in SPARS2 for detecting interfaces should increase significantly, which would severely limit the practicality of the method in high dimensions. The space requirements for the book-keeping information, hwoever, is also a function of the number of nodes. Since the resulting graphs are sparse by nature and contain a small number of nodes, this does keep the space requirements for the book-keeping information relatively low. This allows to identify a conservative bound on the number of book-keepeing entries needed as $O(n \cdot k^2)$, where $n$ is the number of nodes and $k$ is an upper bound on the number of neighbors each node has. This is because at worst, each of the $n$ nodes in the graph must potentially store interface information between each pair of neighbors. This is obviously higher than the storage requirements for edges in PRM*, which is $O(n \cdot k)$, where $k$ is in the order of $O(logn)$. Neverhtheless, the number of nodes in the sparse representation is significantly lower than in PRM*. Furthermore, the following section is going to show that the average valence of the nodes in the sparse roadmaps is actually significantly smaller than that of PRM*, which implies a lower $k$ value as well.

## 6 Simulations

This section provides a series of experimental results, which validate the analysis and the practicality of the proposed sparse roadmap spanners.

### 6.1 Experimental Setup

Experiments were performed on a cluster of the Computer Science department of Rutgers University, which is composed of IBM e-server xSeries 355 machines with 2.8 GHz Intel Xeon quad-core processors and 2GB of memory each. The methods were tested using the Open Motion Planning Library (OMPL) (Şucan et al., 2012) and the environments "Maze" ($SE(2)$), "Pegs" ($SE(2)$), "Abstract" ($SE(3)$) and "Beam Site" ($SE(3)$), shown in Figure 11. Both variations of the proposed framework, SPARS (Section 4.1) and SPARS2 (Section 4.2) were evaluated, and

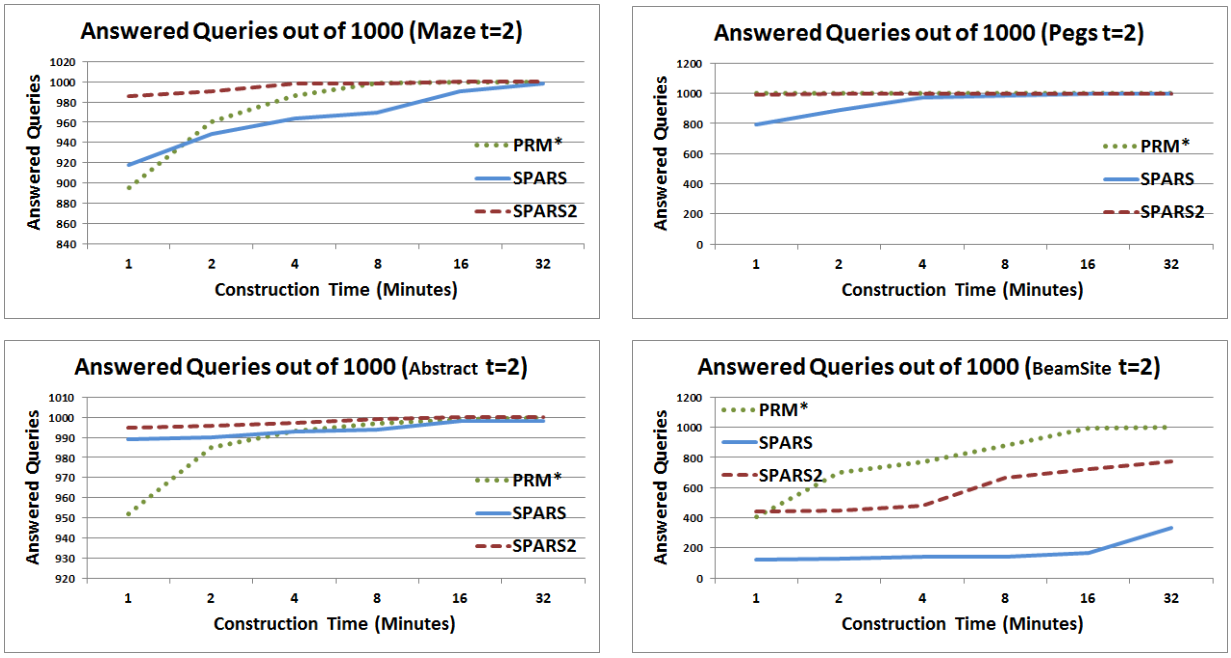


Figure 12: The number of successful queries out of 1000 after $\{1, 2, 4, 8, 16, 32\}$ minutes of roadmap construction time. The expected trend is that as iterations increase, all planners should be able to answer all 1000 queries.

compared against PRM*. Runs were tested with the parameters $\delta = 0.5$, $\Delta = 15$ fixed, and for varying values of the stretch factor $t$ $(2, 3, 5, 9)$. The parameter $k$ was selected to be two times the dimensionality of the $C$-space, e.g., 6 for the $SE(2)$ challenges. The parameter $M$ was removed so that it would be possible to observe the behavior of the algorithms as a function of run time. Statistics were collected after 1, 2, 4, 8, 16, and 32 minutes of construction time for the roadmap spanner methods and compared to statistics collected by PRM* for the same time intervals. An effort was made to extend these experiments up to 64 minutes but the space requirements for PRM* did not allow lengthier experiments. If not otherwise specified, the following graphs show results for a stretch factor $t = 2$. In general, the effects of the stretch factor were small in the chosen environments as the first three criteria often correspond to more than 75% of the nodes added in the roadmap. For each combination of environment, algorithm and parameters, 20 experiments were performed and their output was averaged in order to acquire the following statistics.

## 6.2 Query Success Ratio

It is first important to consider the success ratio of the algorithms over time. Given the probabilistically complete nature of the methods, it is expected that they should be able to eventually solve all problems. Figure 12 shows the number of successfully answered queries out of 1000 as a function of construction time. It is expected that as construction time increases, that the number of successful queries to increase to 1000. In general, the algorithms tend to converge. Interestingly, the graphs for the "Maze" and "Abstract" environments show that early in the execution of the algorithms, PRM* is unable to answer many queries, while SPARS and SPARS2 answer more. The reason for this can be that PRM* converges slower to a connected structure. This is reasonable as early on SPARS and SPARS2 quickly add nodes for coverage and connectivity purposes with an average iteration cost that is lower (edges are bounded by $\Delta$, fewer connections are attempted than with PRM* and no interfaces are detected early on). Nevertheless, this is not a globally consistent behavior, as in the more complex "Beam Site" environment SPARS and SPARS2 answer fewer queries than PRM*. A possible explanation is the choice of parameters $\delta$ and $\Delta$, as the effects of these parameters are still not clear. The path quality statistics in the rest of this section are computed only over successfully solved queries.

## 6.3 Path Quality

The proposed methods guarantee that solutions to queries are within a bound of the optimal solution to these queries.

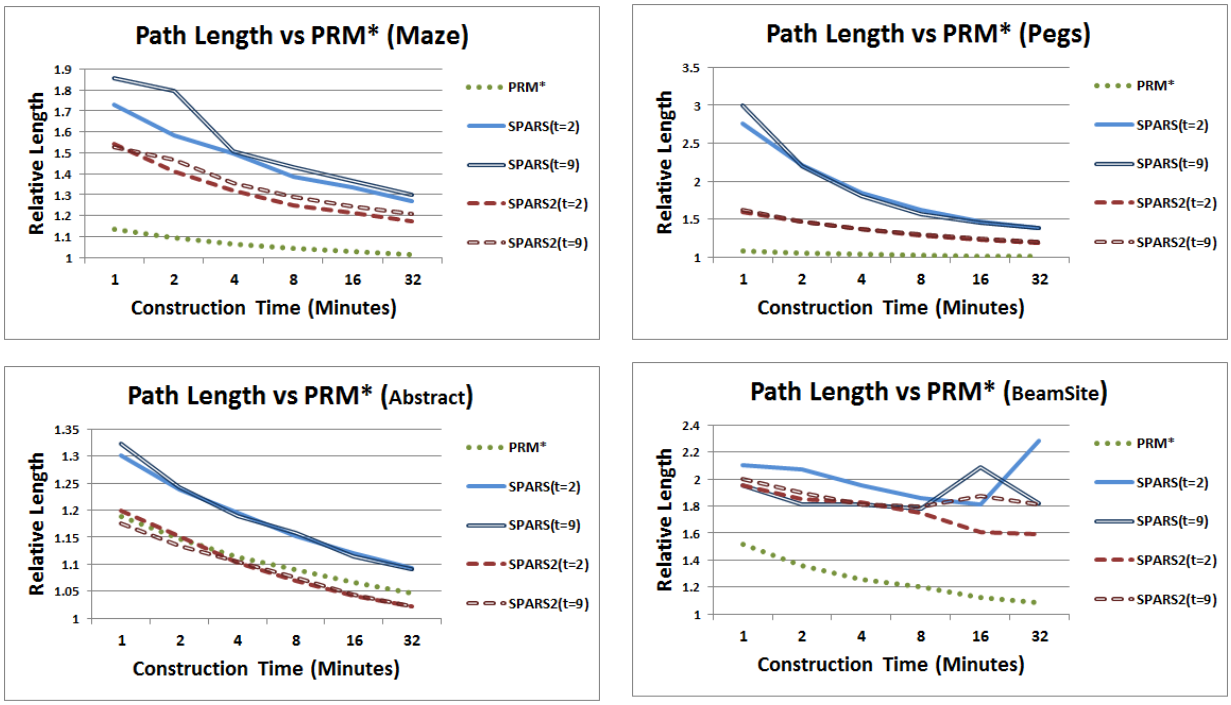


Figure 13: Average path quality for solved queries relative to the best paths found by PRM* after 32 minutes. As time increases, the returned path length should be decreasing. The results for the "Beam Site" environment are affected by the lower success ratio of the spanner algorithms.

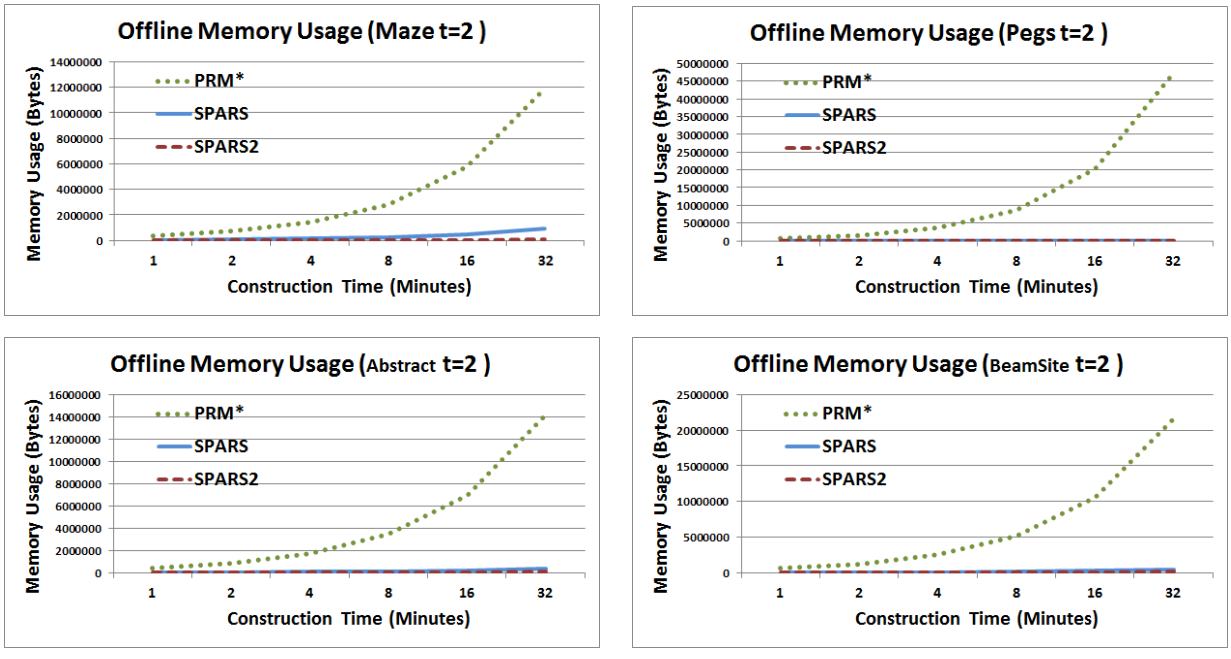


Figure 14: The memory usage of the three algorithms while they are preprocessing the space. Both SPARS and SPARS2 provide significant advantages in terms of memory over PRM*.

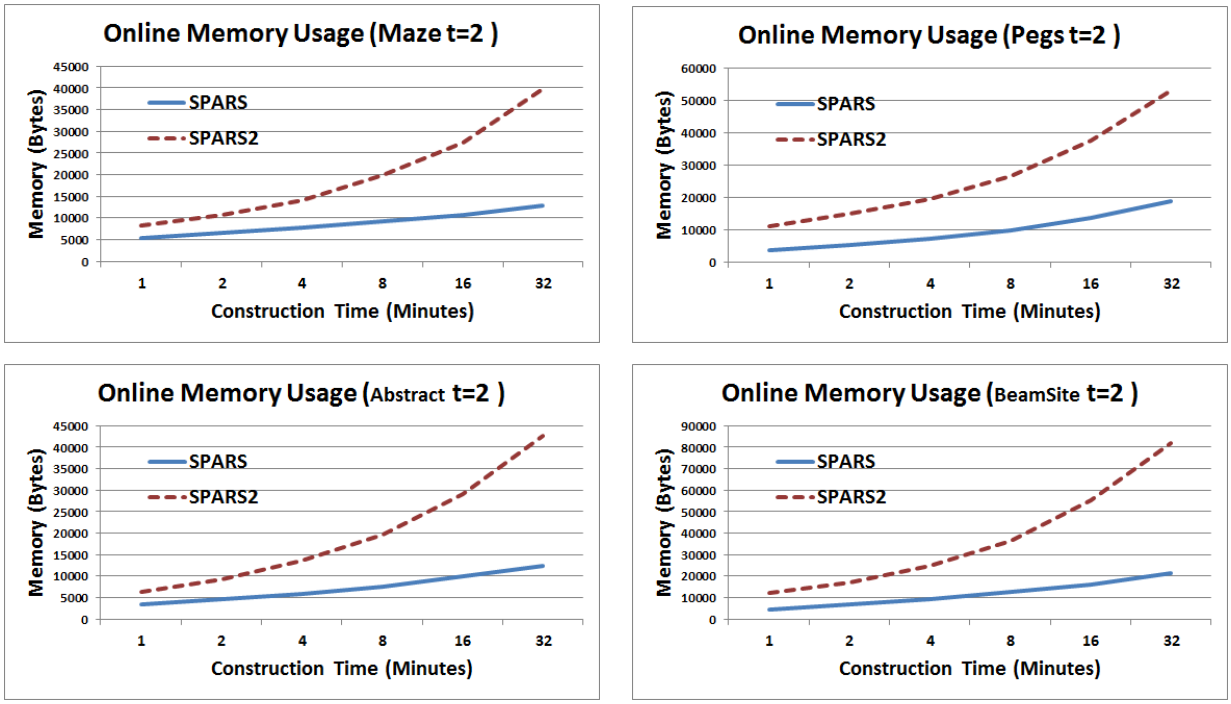


Figure 15: The memory required to store the resulting roadmap. PRM* is omitted as the memory requirements are the same as the offline ones (see previous figure) and significantly higher compared to the spanner methods .

The graphs in Figure 13 show the average path quality returned by each of the planners relative to the best paths returned by PRM* after 32 minutes of execution. In these graphs, a value of 1.0 represents a path which has the same cost as the best PRM* path. The PRM* paths asymptotically converge to the optimum, so they are used as a near approximation of the true optimal path cost. Note that the average PRM* relative length after 32 minutes may still be higher than 1.0 and depends on the consistency of the algorithm in returning the best solution. Interestingly, even after a few minutes of execution, the returned path quality by the spanner algorithms is actually much better than what the theoretical bounds would suggest. In the $SE(2)$ environments the results are very consistent, as time increases all planners return solution paths of decreasing length with PRM* closer to the optimum, SPARS2 following and SPARS with a higher degradation. In certain environments, such as the "Abstract" environment, the spanner methods get to within $110\%$ of the optimal length, whereas the stated bounds are relaxed to as much as $200\%$ or $900\%$ for these experiments. Note that for the same environment, the SPARS2 algorithm is more consistent in returning better quality paths than the PRM* itself. Given the results of Figure 12, the "Beam Site" environment had fewer answered queries, especially for the SPARS algorithm. As new queries are answered, their path lengths are added to the averages and in some cases the averages are shown to increase as construction time increases. Nevertheless, even in this case the relative path length is lower than the theoretical bounds.

## 6.4 Offline Memory Requirements

Figure 14 provides a comparison of the amount of memory used during the roadmap construction step. These results do include any extra bookkeeping information required by the SPARS and SPARS2 algorithms, as well as the cost of maintaining the sparse and/or dense graph. As expected, PRM* uses a significantly higher amount of memory. As was also expected, however, SPARS uses more memory than SPARS2 during this process, as it maintains the dense graph $D$. The result which is encouraging is the significant difference in used memory between SPARS and PRM*. Note, however, that the duration of each iteration in SPARS is much larger than each iteration of PRM*, and as such, the size of the dense graph ends up being smaller than the resulting graph from PRM* given the same amount of preprocessing time. One of the primary objectives of the SPARS2 approach was to reduce the memory footprint during the preprocessing stage

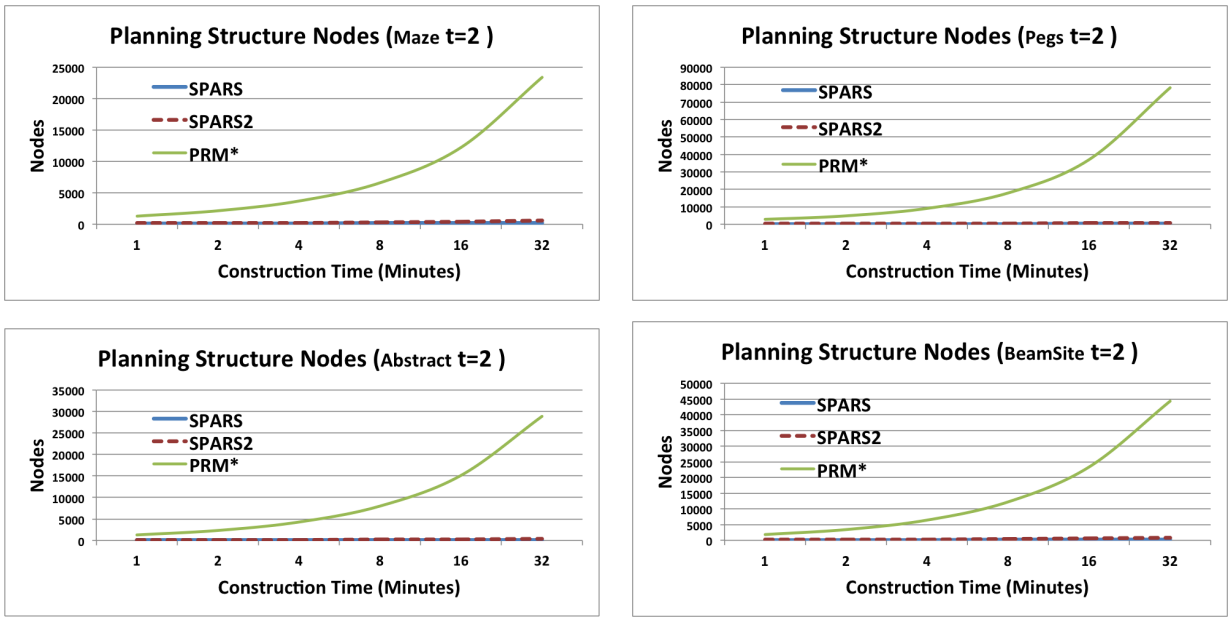


Figure 16: The number of nodes generated and stored for the final query structure of the various methods in the four environments.

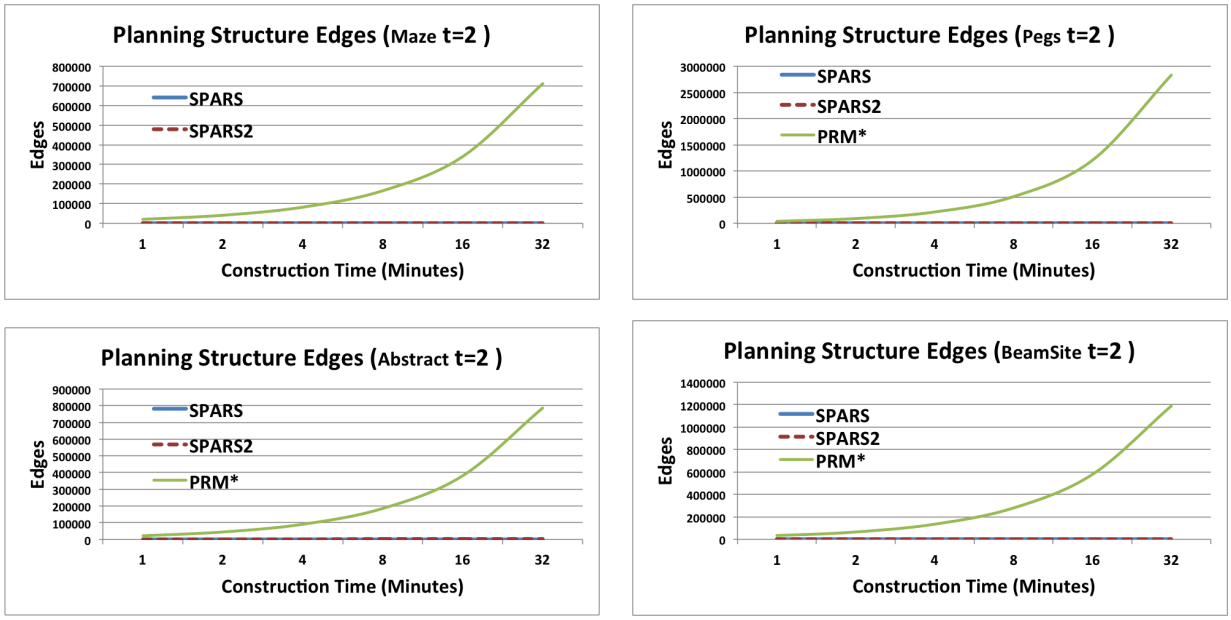


Figure 17: The number of edges stored for the final query structure of the various methods in the four environments.

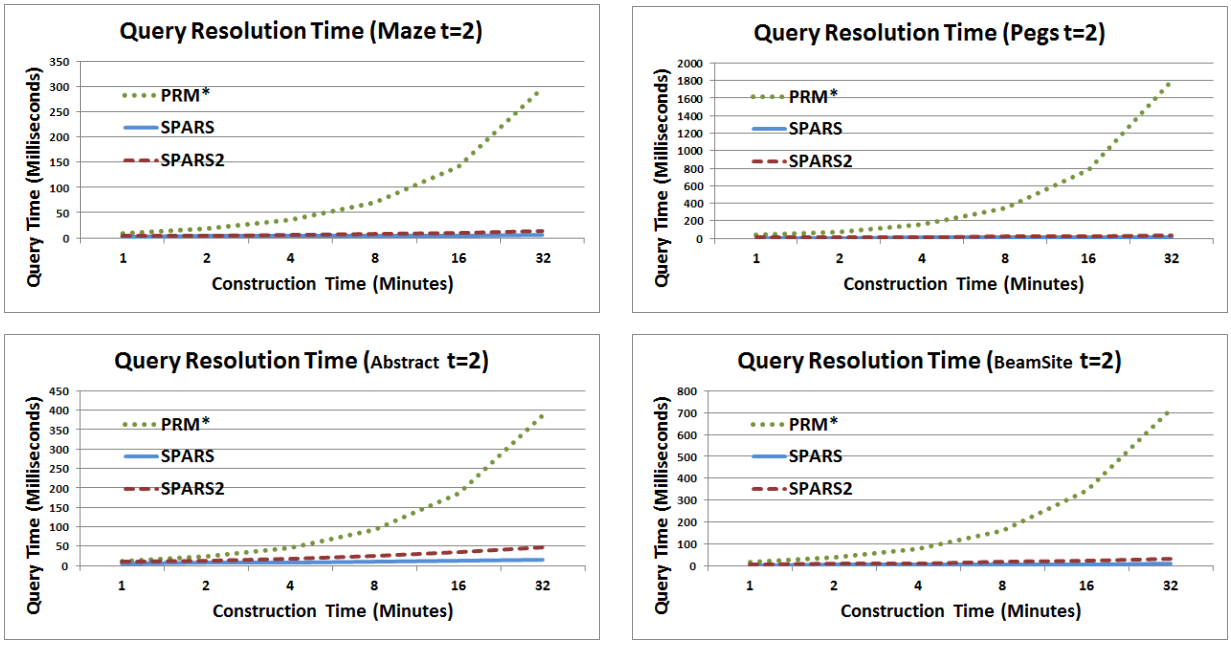


Figure 18: Query resolution time as a function of construction time. It is expected that since the size of the planning structures increases, query resolution time also increases.

while maintaining the guarantees on path quality, and in all cases, SPARS2 takes much less memory than SPARS as the following table suggests:

| Environment | SPARS | SPARS2 |
|---|---|---|
| Maze | 94,0373.4 | 51,125.0 |
| Pegs | 104,453.3 | 54,406.2 |
| Abstract | 332,673.9 | 52,634.6 |
| Beam Site | 386,662.6 | 91,605.6 |

Table 2: Memory requirements in bytes for each spanner method after 32 minutes of computation.

## 6.5 Online Memory Requirements

After preprocessing the space, each algorithm returns a graph which is used for answering queries. Larger graphs will have higher memory requirements for storage and transmission. It is expected that as construction time increases, the size of the resulting graphs will be larger. Typically, the larger the graph, the better the path quality. The results shown in Figure 15 omit the graphs returned by PRM*. This is because the memory requirements are the same as the offline process, and significantly higher than the roadmaps returned by SPARS and SPARS2, which would render the comparison of the last two methods impossible. It is important to note that in every case, SPARS2 is returning larger graphs than SPARS. Note that SPARS performs heavier computation when considering adding nodes for path quality. Multiple $A^*$ searches are performed on an increasingly larger dense graph, and representative information must be preserved for every node in the dense graph.

## 6.6 Graph Nodes and Edges

The online memory requirements of the methods are a direct result of the number of nodes and edges in the resulting graphs, shown in Figures 16 and 17 respectively. Parallel to the online memory requirements, the SPARS and SPARS2 methods use orders of magnitude less nodes and edges in their final query structure.

## 6.7 Query Resolution Time

The amount of time to resolve a query can be very important for certain applications. The query resolution time is directly correlated to the size of the planning structure being queried. Figure 18 shows resulting query times for the approaches, and correlates with Figure 15. Both SPARS and SPARS2 return queries orders of magnitude faster than querying the roadmap returned by PRM*. SPARS2 tends to return very high quality paths relatively early; however, the query times tend to be higher than those returned by SPARS.

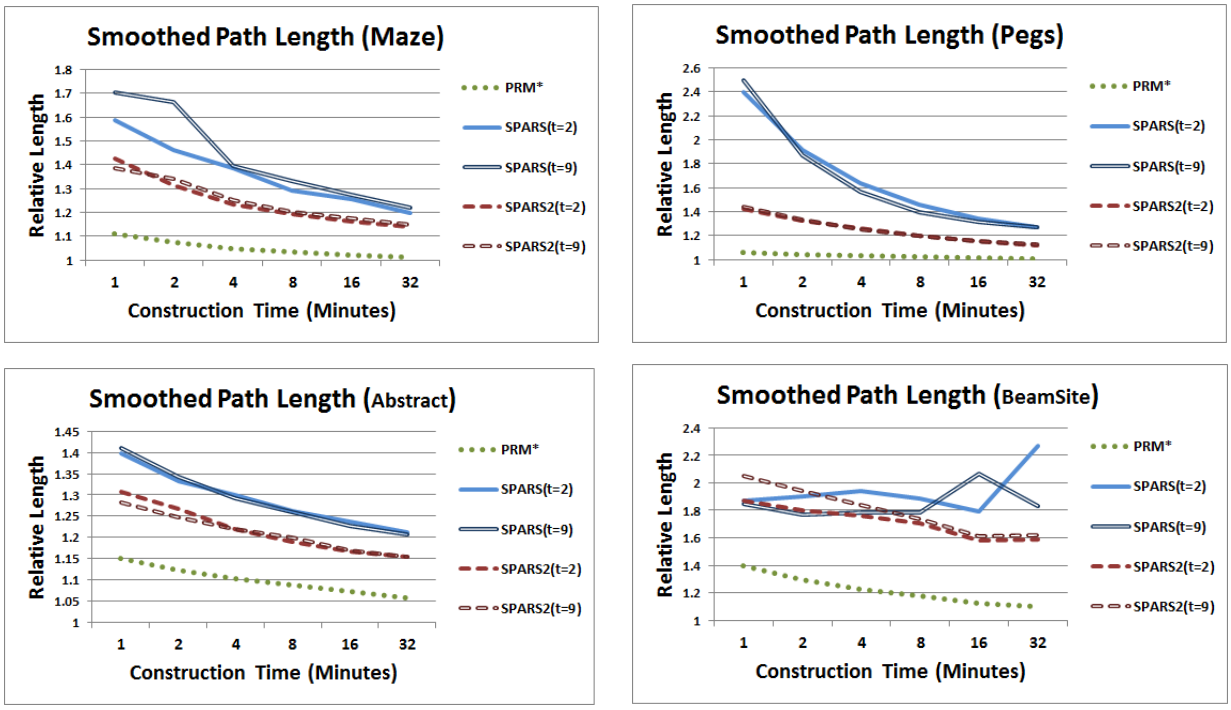


Figure 19: Path quality relative to the best smoothed paths from PRM*.

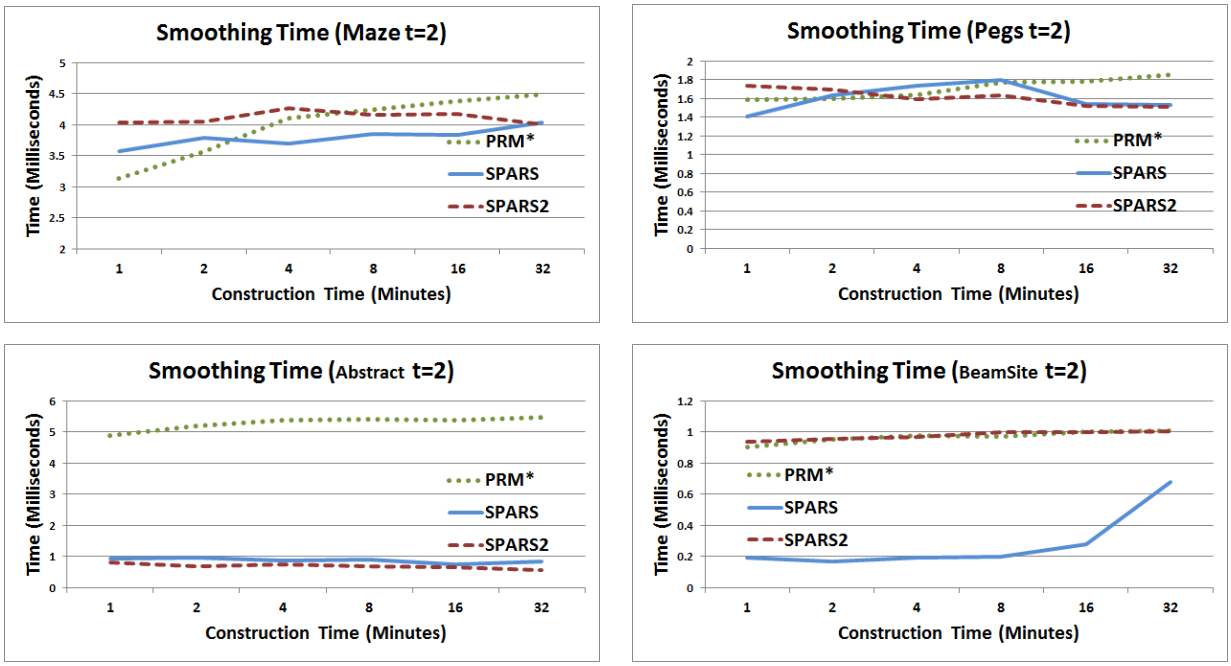


Figure 20: Time spent smoothing queries returned by the planning structure. Higher smoothing times represent greater reductions in path length.

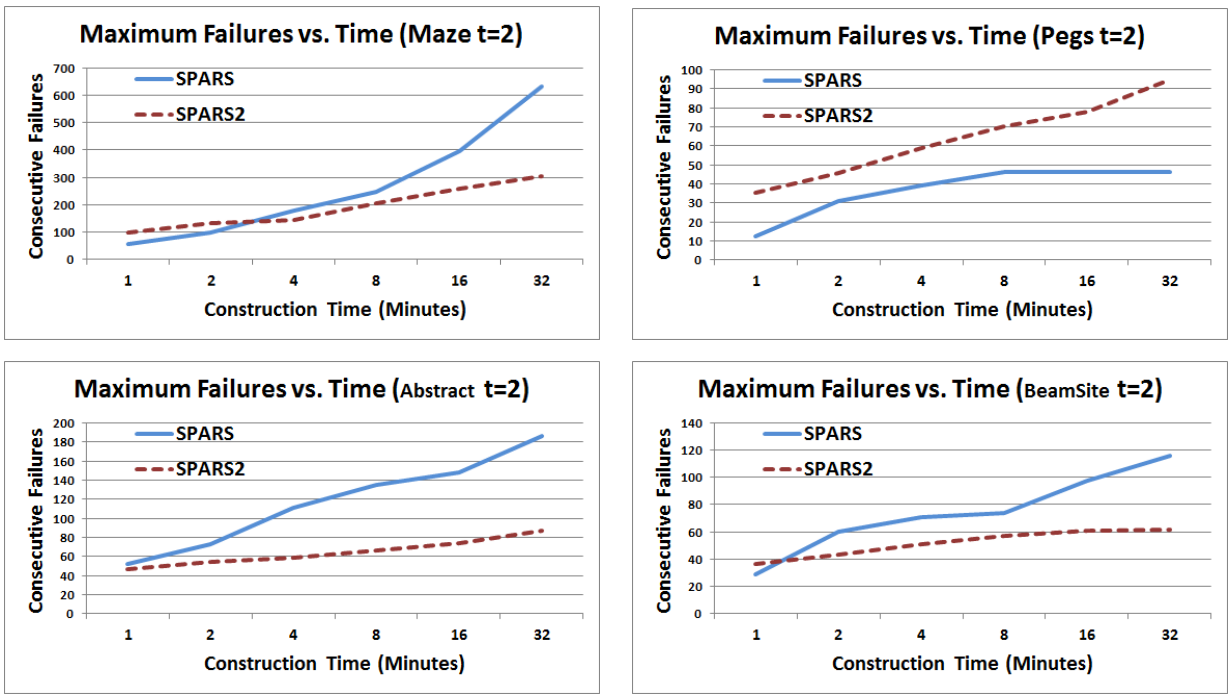


Figure 21: The average highest number of consecutive failures reached by the methods. If parameter $M$ were set to a value lower than the graphs, then the algorithms would have automatically terminated.

## 6.8 Effects of Smoothing

Smoothing can have a significant effect on the resulting path quality. Figure 19 shows the average smoothed solution path length relative to the best smoothed path returned by PRM*. In general, it is expected that PRM* will have less benefit from smoothing than other methods as it already returns paths which are converging to the true optimum. Smoothing provided a significant decrease in relative path length in both $SE(2)$ environments "Maze" and "Pegs". Nevertheless, the effects are less pronounced in the $SE(3)$ environments. In the "Abstract" environment, the smoothing process was advantageous for the PRM* method.

Smoothing also introduces a time overhead as it involves collision checking, and in certain application areas this overhead can be expensive. Figure 20 details the amount of time spent on average for smoothing solutions to the ones shown in Figure 19. In general, all of the methods use roughly the same amount of time for smoothing throughout their execution. The smoothing time is also reflective of how much the path is able to improve relative to the original path cost. In the "Abstract" environment, PRM* is shown to have much larger smoothing time than the other methods. This suggests that the paths returned in this environment could be improved significantly by smoothing. This is why in Figure 19, the relative path quality shown for SPARS and SPARS2 appear worse than before smoothing. In the case of the "Beam Site" environment, the smoothing time of paths returned by SPARS are very small, suggesting that the smoothing was unable to refine these paths very much. This suggests that it is returning poor quality paths in homotopic classes different than the optimum one. It is apparent that the algorithm was still in the process of converging.

## 6.9 Maximum Consecutive Failures

The approach proposes an automated stopping criterion in the form of a threshold for the maximum number of consecutive failures to add a node to the planning structure. Figure 21 shows the rate of growth of the average maximum consecutive failures reached by the methods as they run. As expected and as is desirable, the number of maximum consecutive failures increases over time. In most cases, SPARS reaches a higher number of consecutive failures than SPARS2. A possible explanation is that it is more difficult for the method to identify the need to add dense paths using the dense graph than through a focused sampling process. This correlates with the fact that SPARS returns smaller planning structures than SPARS2.

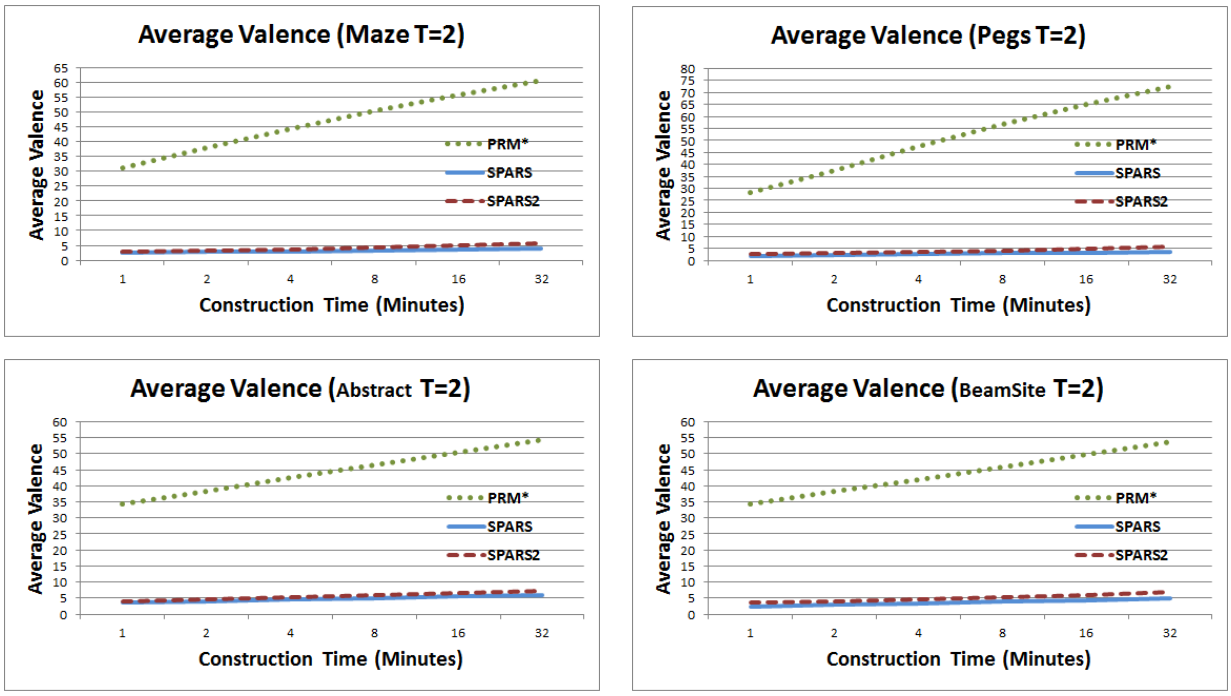


Figure 22: The average valence for the nodes in the resulting planning structure at various points during construction.

## 6.10 Average Node Valence

An interesting aspect of the resulting planning structures to study is the average valence of the nodes in the graph, which relates to the sparsity of the resulting data structure. PRM* requires a certain number of attempts to connect to neighbors, which is a function of the number of nodes in the roadmap, which is reflected in the result. Figure 22 shows how the average valence increases for the methods over time and shows that PRM* builds a much denser graph compared to both SPARS and SPARS2.

## 6.11 Online Memory Use vs. Path Quality

A direct correlation can be shown between the size of the resulting planning structure and the path quality which is returned by the structure. Figure 23 shows this relationship for SPARS and SPARS2 for stretch factors $t = 2, 9$ at 32 minutes of construction time. The intuition is that larger structures should return better quality paths in general, as they are a closer approximation of the underlying exhaustive graph of $C_{\text{free}}$. The results, especially in the "Maze" environment, demonstrate this trade-off between memory use and path quality. Note that if these graphs were to include the PRM* algorithm, the corresponding data point would appear orders of magnitude higher on the $y$ axis. For example, Figure 24, shows the online memory use over time for the methods, including PRM*. Note that the SPARS techniques do not necessarily reach the same level of path quality as PRM*, with the "Abstract" environment being the exception. Note however that when SPARS and SPARS2 do approach the same relative path length, the methods use far less memory than PRM*.

## 6.12 Query Time vs. Path Quality

Considering the correlation already drawn between resulting planning structure size and query resolution time, it is expected that there would also be a correlation between query time and path quality given the results shown in the previous subsection. The results in Figure 25 show this correlation, which is very similar to that provided in Figure 23.

## 6.13 Problem of Increasing Complexity

It is not straightforward from the above experiments to determine how well the SPARS methods extend to more difficult problem instances. A straightforward method for increasing problem difficulty is to increase the size of the robot (or equivalently grow the obstacles), so as to create a more constrained free space. This section examines the "Abstract" environment using an increasingly larger

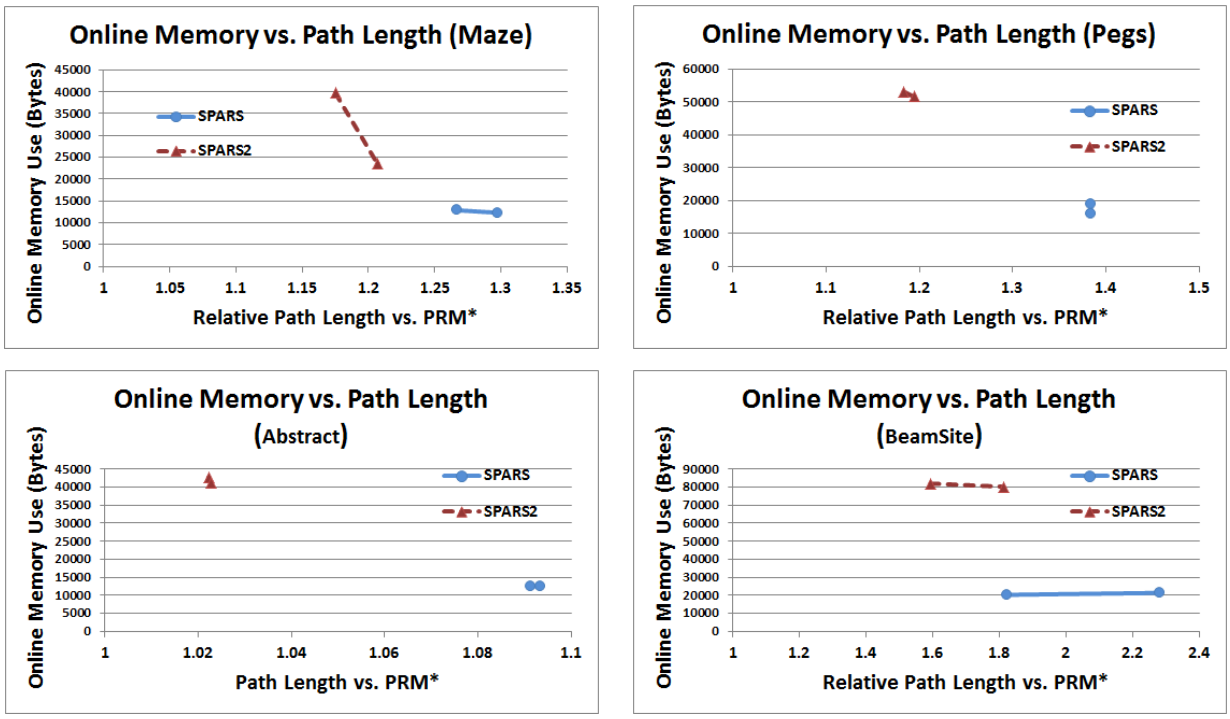


Figure 23: A comparison of online memory usage relative to the average returned path quality for the resulting planning structures. Larger structures should return better quality paths. Data points for PRM* are omitted for scaling reasons, but are given as $(1.0140, 11946512.8)$ (Maze), $(1.0080, 47172614.4)$ (Pegs), $(1.0467, 14198567.2)$ (Abstract), and $(1.0852, 21512684.8)$ (BeamSite).

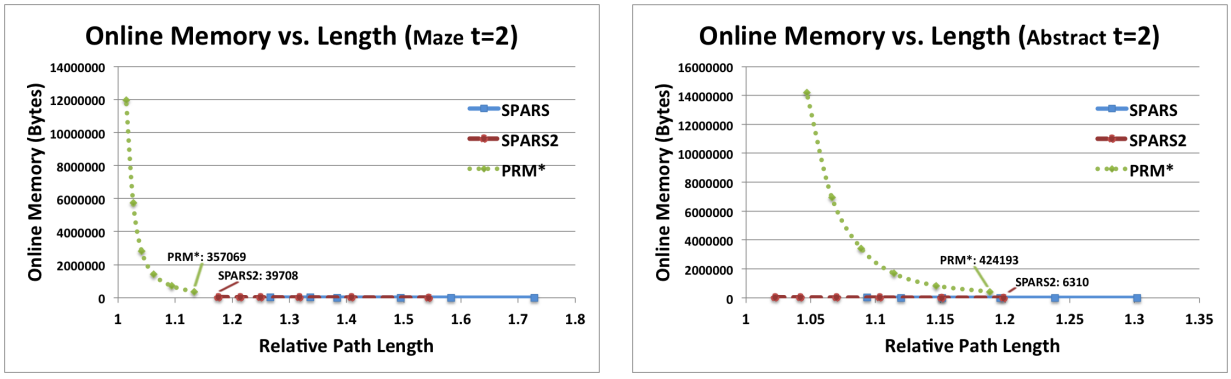


Figure 24: A comparison of online memory used against returned path quality through time for stretch factor $t = 2$. Values for the online memory use are highlighted for the lowest memory cost PRM* graph and it's closest neighbor from the SPARS algorithms.

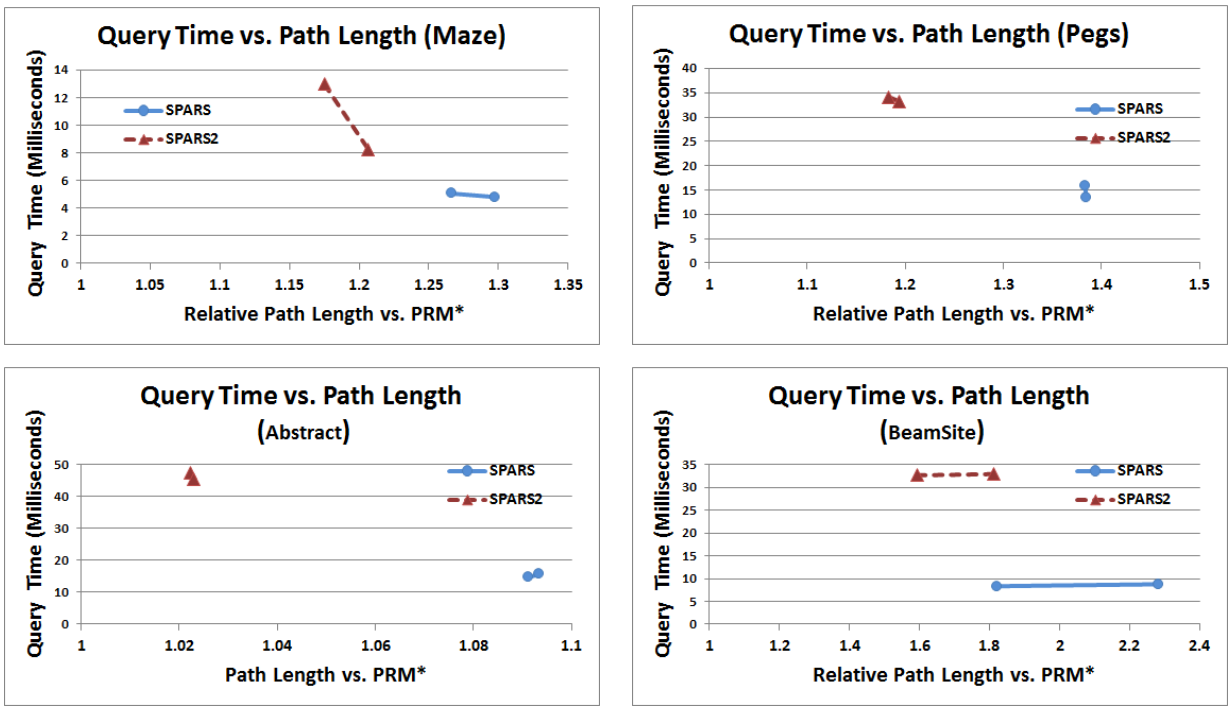


Figure 25: A comparison of query time against average path quality. It is expected that paths of high quality take longer to query than those of low quality.

L-shaped robot. Figures 26 - 29 show interesting aspects of the methods when increasing problem difficulty.

Figure 26 shows the relative path length of SPARS and SPARS2 compared to PRM*. The figures indicate that as the problem difficulty increases, both the SPARS and SPARS2 methods show an increase in path length relative to PRM*. Figure 27 shows that the offline memory requirements in more complex environments stays relatively the same, with the SPARS methods using slightly more memory and PRM* using slightly less memory as problem complexity increases. As the problem difficulty increases, there are fewer collision-free configurations for PRM* to include in the roadmap. The sparse roadmaps need to include a higher number of nodes to solve the problem. Nevertheless, the difference is not significant over the various problem instances.

Figure 28 shows the trade-off between memory use and path quality for increasingly difficult problem instances. The trend shows the data points moving up and to the right, indicating that more difficult problem instances require more memory while also yielding lower-quality paths. Finally, Figure 29 indicates how many queries are answered for varying difficulty of problems. As expected, the number of answered queries decreases for more difficult problems. An interesting data point is the intersection between the PRM* graph and those of SPARS and SPARS2 methods. These data indicate that the point of intersection may be coming later for more difficult problems, i.e., it takes longer for PRM* to solve a larger percentage of queries.

## 6.14 Types of Nodes Added over Time

Both the SPARS and SPARS2 methods add nodes to the graph only if they satisfy one out of four criteria. It is interesting to see what criteria are satisfied through the run of the algorithm. Figure 30 shows averaged data for nodes added to SPARS2 over time. Note that the algorithm starts by quickly covering and connecting the space, followed by a reduction in node addition for these purposes as the sampling process works towards enough sample saturation to begin detecting interfaces.

One question was whether the fourth criterion, which relates to path quality and the spanner property, contributes significantly to the number of nodes added to the roadmap. The graphs in Figure 30 indicate that this is indeed the case. Were these types of nodes in fact rarely added to the graph, a simplified version of the algorithm which omits this final, complicated criterion could be potentially sufficient to provide the desired behavior, i.e., a

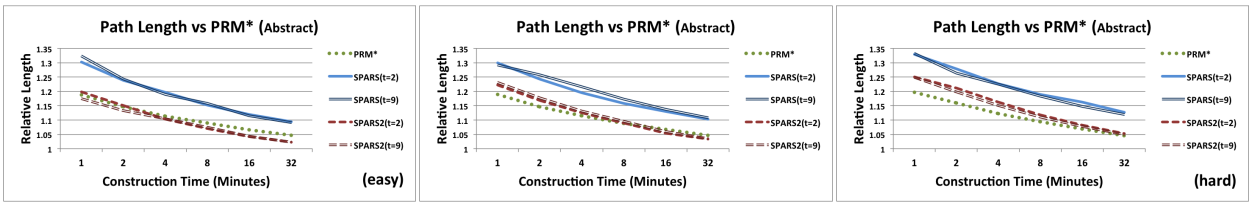


Figure 26: Relative path length to best path found for increasing problem difficulty from left to right. It is surprising that SPARS2 may even return better path quality than PRM* in easy problems. For harder instances, PRM* will be able to return better paths for longer construction times as expected.

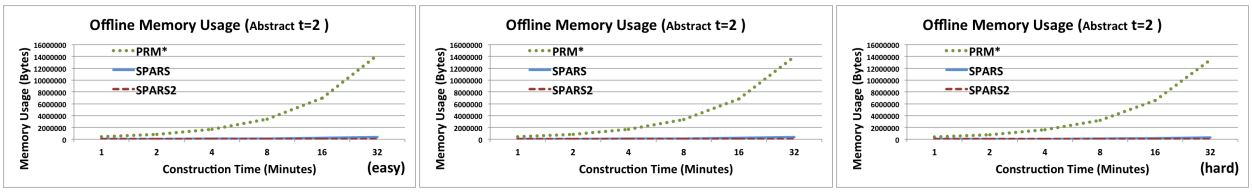


Figure 27: Memory usage during offline preprocessing for the various methods for problems of increasing difficulty from left to right. Memory usage remains relatively the same, though PRM* reduces its memory requirements, where SPARS and SPARS2 increase their memory requirements. Both results are expected.

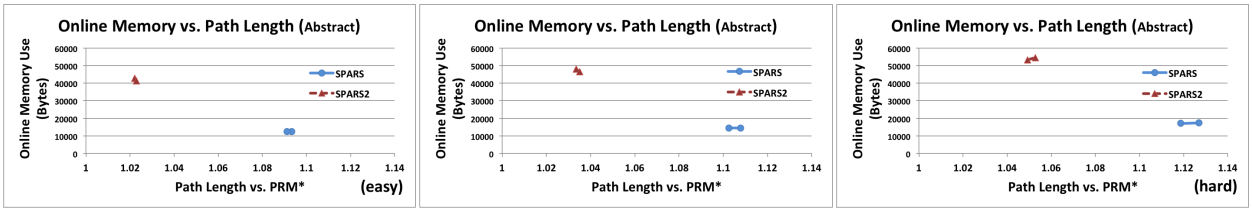


Figure 28: Trade-off between online memory usage and path length for the various methods for problems of increasing difficulty from left to right. Both memory usage and path length increase as the problem becomes more difficult. Data points for PRM* are omitted due to scaling issues. The data points for PRM* for the graphs from left to right are $(1.0467, 14198567.2)$, $(1.0460, 13932344)$, and $(1.0453, 13426218.4)$.

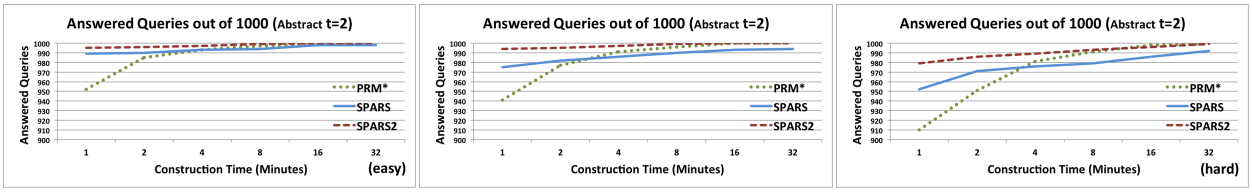


Figure 29: Successfully answered queries for the various methods for problems of increasing difficulty from left to right. These graphs indicate the increase in difficulty for solving the problem. In all cases, PRM* has the worst performance early on and then converges to the success ratio of SPARS2.

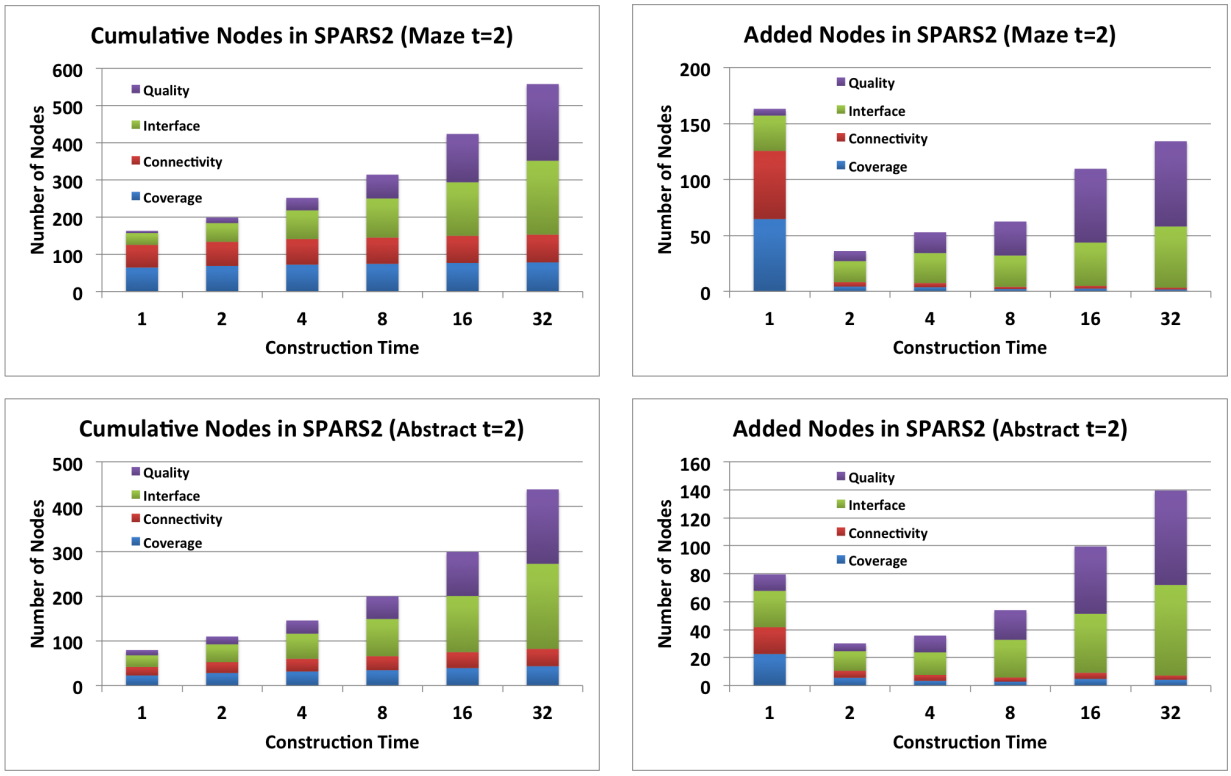


Figure 30: Average nodes added due to each criterion in the SPARS2 algorithm for the Maze and Abstract environments. Left: cumulative node totals at given time intervals; Right: node additions between intervals.

sparse representation that can quickly answer queries with path of sufficient quality. This simplified algorithm would not be providing the desired theoretical properties, but potentially it could still be a practical solution as it would reduce the amount of bookkeeping. The indication, however, from these graphs is that because nodes for upholding the spanner property are a significant percentage of the total number of nodes, removing this criterion would significantly affect path quality.

# 7 Discussion

A framework for generating sparse roadmap spanners is presented in this work as a way to solve path planning problems in continuous configuration spaces using compact data structures, while providing the property of asymptotic near-optimality. Two variants of this framework are highlighted, SPARS and SPARS2. The resulting planning structures from both methods are orders of magnitude sparser and smaller than the corresponding asymptotically optimal structures, while maintaining good quality paths. This results in significantly shorter query resolution times. The resulting graph spanners are shown to provide high quality paths that come much closer to optimal ones than what the theoretical bounds specify. SPARS2 also reduces memory requirements upon construction of the roadmap versus SPARS while returning even better quality paths at the cost of a small increase in the size of the final roadmap. It does so by removing the dependence on maintaining an explicit, dense graph representation of the space. Instead, it relies on properties of visibility that can be computed through localized sampling and smoothing processes to provide the same guarantees.

There are many directions to investigate into the future on this topic: (i) It is interesting to study similar near-optimality challenges in the context of graphs with directed edges, which is a necessary requirement for systems with constraints in their motion. The current line of reasoning relies on being able to directly connect configurations using a bidirectional steering method, which is often unavailable for systems with dynamics. (ii) An important step for the work is to show whether the planning structure converges to a finite-sized roadmap. Showing that the probability of adding nodes goes to 0 is a step in this direction but does not guarantee that the desired

properties are provided by finite graphs. (iii) An exciting development would be to compute a confidence value representing what volume of the optimum paths in the space are covered by the planning structure after a finite time execution instead of studying the asymptotic case. This effort could lead to a stopping criterion for the algorithm that would allow the computation in finite time of probably near-optimal paths with a confidence value. (iv) As the method is able to return solutions within a bound of optimal paths, it would be interesting to show whether the method can guarantee that it finds paths in important homotopic classes of the space, and to see how it compares to methods which attempt to identify these classes explicitly (Jaillet and Simeon, 2006). (v) It is unknown how to select parameters $t$ and $\Delta$ to attain an expected average path degradation. Furthermore, it is interesting to evaluate how the results depend on other parameters of the algorithm, such as $\delta$ and $M$. (vi) Finally, many ideas can be exploited to improve computational efficiency, such as using tools which return distance to obstacles to reduce collision-checking calls and possibly quickly identifying nodes which should or should not be considered for addition to the graph.

**Funding**

This work has been supported by NSF award CNS 0932423. Any opinions, findings and conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect the views of the sponsor.

**Acknowledgments**

The authors would like to thank the anonymous WAFR and IJRR reviewers and editors who provided insightful comments and suggestions related to this work.

# References

P. Agarwal. Compact Representations for Shortest-Path Queries. In *IROS Workshop on Progress and Open Problems in Motion Planning*, September 2011.

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

M. S. Apaydin, D. L. Brutlag, C. Guestrin, D. Hsu, J.-C. Latombe, and C. Varm. Stochastic Roadmap Simulation: An Efficient Representation and Algorithm for Analyzing Molecular Motion. *Journal of Computational Biology*, 10:257–281, 2003.

J. Barraquand and J.-C. Latombe. Robot Motion Planning: A Distributed Representation Approach. 10(6):628–649, December 1991.

S. Baswana and S. Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures and Algorithms*, 30(4):532–563, July 2007.

R. Bohlin and L. Kavraki. Path Planning Using Lazy PRM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 1, pages 521–528, San Francisco, CA, April 2000.

V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian Sampling Strategy for Probabilistic Roadmap Planners. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1018–1023, Detroit, MI, May 1999.

M. Branicky, S. LaValle, K. Olson, and L. Yang. Quasi-Randomized Path Planning. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1481–1487, Seoul, Korea, May 2001.

R. Brooks and T. Lozano-Pérez. A Subdivision Algorithm in Configuration Space for Findpath with Rotation. pages 799–803, 1983.

J. Canny. *The Complexity of Robot Motion Planning*. PhD thesis, MIT, Cambridge, MA, 1988.

H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston, MA, 2005.

I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, to appear 2012.

A. Dobson and K. E. Bekris. Improving Sparse Roadmap Spanners. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2013 - submitted.

A. Dobson, T. D. Krontiris, and K. E. Bekris. Sparse Roadmap Spanners. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Cambridge, MA, June 2012.

M. Foskey, M. Garber, M. Lin, and D. Manocha. A Voronoi-based Hybrid Motion Planner. In *IEEE/RSJ Intern. Conf. on Intelligent Robots and Systems (IROS)*, 2001.

R. Geraerts and M. H. Overmars. A Comparative Study of Probabilistic Roadmap Planners. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V*, pages 43–58. Springer-Verlag, 2003.

R. Geraerts and M. H. Overmars. Creating High-Quality Roadmaps for Motion Planning in Virtual Environments. In *IROS*, pages 4355–4361, Beijing, China, October 2006.

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*, October 1999.

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.

D. Hsu, J.-C. Latombe, and R. Motwani. Path Planning in Expansive Configuration Spaces. *Int. Journal of Computational Geometry and Applications*, 9(4-5):495–512, 1999.

D. Hsu, T. Jiang, J. Reif, and Z. Sun. The Bridge Test for Sampling Narrow Passages with Probabistic Roadmap Planners. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 4420–4426, Taipei, Taiwan, September 2003.

Y. K. Hwang and N. Ahuja. A Potential Field Approach to Path

Planning. 8(1):23–32, February 1992.

L. Jaillet and T. Simeon. Path Deformation Roadmaps. In *WAFR*, New York City, NY, July 2006.

M. Kallman and M. Mataric. Motion Planning Using Dynamic Roadmaps. In *ICRA*, volume 5, pages 4399–4404, New Orleands, LA, April 2004.

S. Kambhampati and L. S. Davis. Multiresolution Path Planning for Mobile Robots. 2(3):135–145, September 1986.

S. Karaman and E. Frazzoli. Incremental Sampling-based Algorithms for Optimal Motion Planning. In *RSS*, Zaragoza, Spain, 2010.

S. Karaman and E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. *IJRR*, 30(7):846–894, June 2011.

L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE TRA*, 12(4):566–580, 1996.

L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe. Analysis of Probabilistic Roadmaps for Path Planning. *IEEE TRA*, 14 (1):166–171, 1998.

O. Khatib. Real-time Obstacle Avoidance for Manipulators and Mobile Robots. 5(1):90–98, 1986.

D. Koditschek. Robot Planning and Control via Potential Functions. In *The Robotics Review 1*, pages 349–367. MIT Press, 1989.

J. J. Kuffner. Cloud-Enabled Robots. In *IEEE-RAS International Conference on Humanoid Robotics*, Nashville, TN, 2010.

A. M. Ladd and L. E. Kavraki. Measure Theoretic Analysis of Probabilistic Path Planning. *IEEE TRA*, 20(2):229–242, April 2004.

F. Lamiraux and L. E. Kavraki. Planning Paths for Elastic Objects Under Manipulation Constraints. *International Journal of Robotics Research*, 20(3):188–208, 2001.

F. Lamiraux and J.-P. Laumond. On the Expected Complexity of Random Path Planning. pages 3306–3311, 1996.

J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

S. M. LaValle and J. J. Kuffner. Randomized Kinodynamic Planning. *IJRR*, 20:378–400, May 2001.

P. Leven and S. Hutchinson. Using manipulability to bias sampling during the construction of probabilistic roadmaps. *IEEE Transactions on Robotics and Automation*, 19(6):1020–1026, 2003.

Y. Li and K. E. Bekris. Learning Approximate Cost-to-Go Metrics To Improve Sampling-based Motion Planning. In *IEEE ICRA*, Shanghai, China, 9-13 May 2011.

T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, pages 108–120, 1983.

J. D. Marble and K. E. Bekris. Computing Spanners of Asymptotically Optimal Probabilistic Roadmaps. In *IEEE/RSJ IROS*, San Francisco, CA, September 2011a.

J. D. Marble and K. E. Bekris. Asymptotically Near-Optimal is Good Enough for Motion Planning. In *ISRR*, Flagstaff, AZ, August 2011b.

J. D. Marble and K. E. Bekris. Towards Small Asymptotically Near-Optimal Roadmaps. In *IEEE ICRA*, Minnesota, MN, May 2012.

O. Nechushtan, B. Raveh, and D. Halperin. Sampling-Diagrams Automata: a Tool for Analyzing Path Quality in Tree Planners. In *WAFR*, Singapore, December 2010.

C. Nielsen and L. E. Kavraki. A Two-Level Fuzzy PRM for Manipulation Planning. In *IEEE/RSJ IROS*, pages 1716–1722, Japan, 2000.

D. Nieuwenhuisen and M. H. Overmars. Using Cycles in Probabilistic Roadmap Graphs. In *IEEE ICRA*, pages 446–452, 2004.

D. Peleg and A. Schäffer. Graph Spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

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 TRA*, 21(4):587–608, 2005.

B. Raveh, A. Enosh, and D. Halperin. A Little More, a Lot Better: Improving Path Quality by a Path-Merging Algorithm. *IEEE TRO*, 27(2):365–370, 2011.

J. H. Reif. Complexity of the Generalized Mover's Problem. pages 421–427, 1979.

E. Rimon and D. Koditschek. Exact Robot Navigation Using Artificial Potential Functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, Oct. 1992.

G. Sánchez and J.-C. Latombe. On Delaying Collision Checking in PRM Planning: Application to Multi-Robot Coordination. *International Journal of Robotics Research*, 21(1):5–26, 2002.

E. Schmitzberger, J. L. Bouchet, M. Dufaut, D. Wolf, and R. Husson. Capture of Homotopy Classes with Probabilistic Roadmap. In *IEEE/RSJ IROS*, pages 2317–2322, 2002.

F. Schwarzer, M. Saha, and J.-C. Latombe. Adaptive Dynamic Collision Checking for Single and Multiple Articulated Robots in Complex Environments. *IEEE Transactions on Robotics*, 21(3):338–353, 2005.

T. Simeon, J.-P. Laumond, and C. Nissoux. Visibility-based Probabilistic Roadmaps for Motion Planning. *Advanced Robotics Journal*, 41(6):477–494, 2000.

S. Sundaram, I. Remmler, and N. M. Amato. Disassembly Sequencing Using a Motion Planning Approach. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1475–1480, Washington, D.C., May 2001.

P. Švestka. *Robot Motion Planning using Probabilistic Road Maps*. PhD thesis, Utrecht University, the Netherlands, 1997.

S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1024–1031, Detroit, MI, May 1999.