University of Nevada, Reno

Provably Asymptotically Near-Optimal Motion Planning with Sparse Data Structures

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
with a major in Computer Science and Engineering.

by

Andrew Dobson

Dr. Kostas E. Bekris, Thesis Advisor

August 2012

University of Nevada, Reno
Statewide • Worldwide

THE GRADUATE SCHOOL

We recommend that the thesis prepared under our supervision by

Andrew Dobson

entitled

Provably Asymptotically Near-Optimal Motion Planning with Sparse Data Structures

be accepted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Dr. Kostas E. Bekris, Ph.D., Advisor

Dr. Monica Nicolescu, Ph.D., Committee Member

Nancy LaTourrette, M.S., Committee Member

Dr. Thomas Quint, Ph.D., Graduate School Representative

Dr. Marsha Read, Ph.D., Associate Dean, Graduate School

August 2012

Asymptotically optimal planners, such as PRM*, guarantee that solutions approach optimal as iterations increase. Roadmaps with this property, however, may grow too large. If optimality is relaxed, asymptotically near-optimal solutions produce sparser graphs by not including all edges. The idea stems from graph spanner algorithms, which produce sparse subgraphs that guarantee near-optimal paths. Existing asymptotically optimal and near-optimal planners, however, include all sampled configurations as roadmap nodes. Consequently, only infinite graphs have the desired properties. This work proposes an approach that provides the following asymptotic properties: (a) completeness, (b) near-optimality and (c) the probability of adding nodes to the spanner roadmap converges to zero as iterations increase. Thus, the method suggests that finite-size data structures might have near-optimality properties. The method brings together ideas from various planners but deviates from existing integrations of PRM* with graph spanners. Simulations for rigid bodies show that the method indeed provides small roadmaps and results in faster query resolution. The rate of node addition is shown to decrease over time and the quality of solutions satisfies the theoretical bounds. Smoothing provides a more favorable comparison against alternatives with regards to path length.

Acknowledgements

This work has been supported by:

- UNR Reed Graduate Student Scholarship 2010
- The National Science Foundation under grant: CNS-0932423

I would like to thank my advisor Dr. Kostas Bekris, who assisted me in editing and completing my thesis successfully, as well as giving the necessary support and motivation to see it through. I would like to thank my committee for reviewing this thesis and providing insightful feedback. Special thanks to my fellow colleagues from the PRACSYS group for their support.

Contents

List of Figures

Chapter 1 Problem Motivation and Inspiration

Sampling-based roadmaps preprocess a configuration space (C-space) to answer multiple path planning queries online [11]. They are practical solutions for challenging, relatively high-dimensional instances. Traditionally, the focus has been on probabilistically complete methods that quickly return collision-free paths regardless of their quality. Recent progress has led to asymptotically optimal planners [9], which, however, need to include all configurations as graph nodes to provide optimality. Many applications, however, require small roadmaps that still provide good quality paths. Resource constrained robots can better communicate, store and query smaller, small roadmaps that are computed offline. Interactive games have a small computational budget for path planning and need to quickly load and query a roadmap online [4]. In dynamic environments, small roadmaps with multiple good quality paths are advantageous when roadmap edges must be invalidated on the fly given moving obstacles [8]. This thesis shows that by relaxing optimality to near-optimality, it is possible to return a sparse roadmap that asymptotically converges to near-optimal solutions. The author's definition of a sparse roadmap differs here from a traditional sense of sparsity, where the concern is on the number of edges in the graph (typically, a sparse graph has edges on the order of $O(n)$ edges) in that the focus is on reducing the number of nodes in the graph, as this work aims to provide sparse representations of continuous spaces. The proposed algorithm, SPARS, will eventually stop adding nodes to the roadmap, which is the first step in showing that a finite-sized data structure can capture the properties of a continuous space in such a way that all paths through

this structure are within a bound of all optimal paths through the continuous space.

1.1 Sampling-based Roadmaps

The first popular method for building a roadmap using sampling was the Probabilistic Roadmap Method (PRM) [11]. The algorithm samples a point in $\mathtt{C}_{free}$, the collision-free part of the configuration space, C, and adds it as a node. It then tries to connect it to the k-closest neighbors (k-PRM) or those within a δ-ball (δ-PRM) using a local planner. The local planner returns paths, which are straight-lines in the configuration-space. If the path is in $\mathtt{C}_{free}$, an edge is added. Several variations exist and various adaptations to the path planning problem have been solved with PRM [2, 21, 23, 24, 25]. This success motivated work on the study of the probabilistic completeness guarantees PRM provides [5, 6, 10, 12]. Some variations focus on small roadmaps that provide coverage, connectivity or good path quality [25]. For instance, visibility-based roadmaps reject nodes if not needed for coverage or connectivity [23]. The Useful Cycles approach tests edges for their usefulness in terms of path quality [18] and combined with the Reachability Roadmap Method returns high clearance paths in 2D and 3D C-spaces [4]. Smoothing can improve solutions and algorithms exist that produce roadmaps with paths in all homotopic classes and thus deformable to optimal ones [7, 22]. Nevertheless, they build relatively dense roadmaps and smoothing can be expensive for online query resolution. Hybridization graphs combine multiple solutions into a higher quality one [20].

Roadmaps require a steering method that exactly connects two states, which is not available for many dynamical systems. Tree-based planners, such as RRT [13] and Expansive Spaces [6], can solve problems with dynamics and already return

sparse graphs. Tree-based planners can benefit from roadmaps that return distances between configurations given C-space obstacles [14]. RRT has been shown to converge to a suboptimal solution almost certainly [9, 17].

1.2 Optimality in Sampling-Based Methods

Until recently, the focus of sampling-based motion planning was returning feasible solutions, rather than optimal ones. One way to approach finding optimal paths is to have a planner which identifies the *homotopic* classes of the configuration-space [7]. The author chooses to not take this approach and instead attempts to reach optimality through a different means.

1.2.1 Asymptotic Optimality

Recent work has provided the conditions under which PRM is asymptotically optimal [9]. Asymptotic optimality implies that as more time is invested in constructing the roadmap, the quality of solutions converges to optimal after infinite computation. The analysis indicates that the number of neighbors is the important variable that controls asymptotic optimality [9], as indicated in Fig. 1.1. A simple PRM that connects samples to neighbors within a δ-ball is asymptotically optimal, but results in a dense roadmap. In general, the number of edges added to the roadmap is on the order of $O(n^2)$, where n is the number of nodes in the roadmap. The roadmap's density can be reduced by considering

algorithm	edges	optimal?
δ-PRM	$O(n^2)$	asympt.
k-PRM	$O(kn)$	no
PRM*	$O(n \log n)$	asympt.
k-PRM*	$O(n \log n)$	asympt.
SRS	$O(an^{1+\frac{1}{a}})$	asympt. near
IRS	$O(n \log n)$	asympt. near
IRS2 ($m < n$)	$O(m \log m)$	no

Figure 1.1: PRM variations and asymptotic optimality. n, m: # nodes in the roadmap.

the k-nearest neighbors, which brings the number of edges down to $O(kn)$, but this version is not asymptotically optimal. The PRM* and k-PRM* algorithms rectify this by selecting the minimum number of neighbors required for asymptotic optimality, which is a logarithmic function of the number of nodes , which results in $O(n\log n)$ edges. These methods use a result from graph theory which shows that maintaining optimality in the structure requires this logarithmic number of connections to neighbors. Nevertheless, with all of these algorithms, the number of neighbors still grows with each iteration and the resulting roadmap size is high, as all samples are added as nodes and it is not clear when to stop sampling.

1.2.2 Asymptotic Near-Optimality

A way to return sparser, good-quality roadmaps is to relax the optimality guarantees by utilizing graph spanners [19]. 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. The parameter t is called the *stretch factor* of the spanner. Applying an efficient spanner [3] on the output of k-PRM* resulted in a Sequential Roadmap Spanner (SRS), 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 [16]. Moreover, the path quality degradation in roadmap spanners is quite smaller in practice than the theoretical guarantees. These approaches, however, still include every sample as a roadmap node. Recent work proposed an extension of IRS, called IRS2, which has a similar objective to the current thesis [15]. IRS2

applies the spanner criterion on nodes, i.e., a node is useful if it connects at least two other nodes with a new path that is at least t times shorter than the best alternative. Unfortunately, it is not possible to argue about asymptotic near-optimality by applying the spanner criterion on nodes, nor to show that the algorithm will converge to a finite graph.

1.3 Contribution

This thesis presents the SPArse Roadmap Spanner (SPARS) algorithm, which: (a) is probabilistically complete, (b) can connect any two query points with a path of length:

$$t \cdot c^* + 4 \cdot \Delta, \tag{1.1}$$

where t and Δ are input to the algorithm, c^* is the cost of the optimum path between the query points in $\mathtt{C}_{free}$ of clearance cl, if one exists, and (c) the probability of adding new nodes and edges converges to 0.

SPARS is a step towards addressing the problem of finding a compact representation for answering shortest-path queries in continuous spaces, which has been proposed as an important challenge for the motion planning community [1]. To the best of the author's knowledge, no work can argue for the existence of finite data structures with some form of near-optimality guarantee for continuous path planning. SPARS provides strong evidence that finite data structures with these properties can be created, and that SPARS does so. The method allows for a natural stopping criterion inspired by Visibility PRM [23]. The criterion relates to a probabilistic measure of how close the roadmap is to a solution that provides the desired properties. This

stopping criterion is not obvious for approaches that include all the samples in the roadmap.

In order to compute the spanner, the method builds an asymptotically optimal dense graph. C-space samples are included in the spanner if they are useful for coverage or connectivity purposes or if they improve path quality relative to paths on the dense graph. The algorithm tries a unique approach in that it is based on the identification of boundaries of "visibility" regions of the spanner nodes through the 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. 1.1. The parameters t and Δ can control the sparsity of the resulting roadmap.

Simulations with rigid bodies ($SE(2)$ and $SE(3)$) indicate that the method provides very sparse roadmaps and that the rate of nodes added to the spanner decreases to 0, resulting in very efficient online query resolution times. Moreover, the path solution quality is significantly better than the theoretical bounds given.

Chapter 2 Problem Setup and Notation

The focus of this chapter is to formally define the problem being solved and some relevant notation. Many fundamental properties are given and the functionality of sampling-based methods are discussed in further detail.

2.1 Problem Statement

The configuration-space abstraction casts a robot's position and orientation as a point q in a d-dimensional space. The collision-free part of $\mathtt{C}$ is denoted as $\mathtt{C}_{free}$. This thesis focuses on the $\mathtt{C}$-spaces of planar and rigid body configurations ($SE(2)$, $SE(3)$), but SPARS is applicable if appropriate metric and sampling functions exist.

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

Robust Feasible Paths *A path planning instance (*$\mathtt{C}_{free}$, q_{init}, q_{goal}*) is robustly feasible if there is a cl-robust path that solves it, for a positive clearance* $cl > 0$. *A path* $\pi \in \Pi$ *is cl-robust, if* π *lies entirely in the cl-interior of* $\mathtt{C}_{free}$.

This work aims towards a planner that provides a compact, finite-size data structure for answering shortest-path queries in continuous spaces to return such robust feasible paths. Such a planner must be able to identify which configuration-space

samples are not needed as roadmap nodes. The standard, graph-theoretic formulation of spanners does not allow for the removal of nodes, so this work focuses on not adding nodes rather than attempting to find appropriate means of node removal. An implicit, exhaustive graph $G(V, E)$ over $\mathtt{C}_{free}$ can be defined by taking all the elements of $\mathtt{C}_{free}$ as nodes and collision free paths between them as edges. Then, a "roadmap spanner" is a subgraph $\mathtt{S}(V_{\mathtt{S}} \subset V, E_{\mathtt{S}} \subset E)$ of this implicit, exhaustive graph of the continuous space with three distinct properties:

- All nodes in G are connected with a path in $\mathtt{C}_{free}$ to a node on $\mathtt{S}$ (coverage).
- $\mathtt{S}$ has as many connected components as G (connectivity).
- All shortest paths on $\mathtt{S}$ are no longer than t times the corresponding shortest paths in G (spanner property).

The properties allow for (a) arbitrary query points to connect to the roadmap, (b) paths to exist between any query points through $\mathtt{S}$ that can be connected in $\mathtt{C}_{free}$, and (c) asymptotic near-optimality for query points that lie on $\mathtt{S}$. If the spanner, however, does not contain all $\mathtt{C}_{free}$ points, there is no guarantee regarding path cost.

Asymptotic Near-Optimality with Additive Cost *An algorithm is asymptotically near-optimal with additive cost if, for a path planning problem ($\mathtt{C}_{free}$, 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 the iterations approach infinity.*

This thesis provides evidence that finite-size data structures can provide the above property and presents an algorithm that asymptotically converges to a sparse data structure with this property.

2.2 Algorithmic Modules and Notation

Local Planner *Given $q, q' \in \mathtt{C}$, a local planner computes a local path $\mathtt{L}(q, q')$ connecting both configurations q and q' in the absence of obstacles. A straight-line between q and q' in $\mathtt{C}$ is often sufficient.*

Distance function *The space $\mathtt{C}$ is endowed with a distance function $d(q, q')$ that returns distances between configurations in the absence of obstacles. The method requires that the distance function operates as a metric within a metric space.*

Shortest Paths *A $cl-$robust shortest path in $\mathtt{C}_{free}$ is denoted as $\pi_{\mathtt{C}}$.*

The shortest path between two configurations computed through the roadmap spanner $\mathtt{S}$ is denoted as $\pi_{\mathtt{S}}$.

Chapter 3 Algorithmic description of SPARS

An exposition of the SPARS algorithm is given in greater detail in this chapter. The chapter provides verbal description of the overall algorithm and great detail is provided on the individual steps of the algorithm. The author provides a pseudocode algorithm showing the functionality of SPARS at a low-level.

3.1 High-level Algorithm Description

SPARS builds two graphs in parallel: (a) a "sparse" roadmap $\mathtt{S}(V_{\mathtt{S}}, E_{\mathtt{S}})$ with the desirable properties, returned upon termination, and (b) a "dense" graph $\mathtt{D}(V_{\mathtt{D}}, E_{\mathtt{D}})$, which instead of being the exhaustive graph G, corresponds to the output of δ-PRM and asymptotically stores optimal paths. The notion of sparsity in this work deviates from its standard use in graph theory and is used to denote that graph S contains a small number of nodes, while D asymptotically includes all configurations as nodes.

SPARS operates by sampling configurations in $\mathtt{C}_{free}$ and adding them to D. The algorithm adaptively selects a subset of these samples as nodes in the roadmap spanner S (i.e., $V_{\mathtt{S}} \subset V_{\mathtt{D}}$). There are four methods for promoting a node in D to S which are tested in order: *guards*, *bridges*, *interface nodes*, and *shortcuts*. A *guard* is added whenever it is not covered by the Δ coverage of nodes already in $V_{\mathtt{S}}$. A *bridge* is added whenever it is covered by multiple nodes in $V_{\mathtt{S}}$ which are in disconnected components of S. An *interface node* is added when it reveals the boundary of a voronoi region between two sparse nodes which do not share an edge in $E_{\mathtt{S}}$. A *shortcut* is added

when a path through D is discovered which is significantly shorter than paths which already exist in $\mathtt{S}$.

By construction, there will be no edge in $E_{\mathtt{S}}$ that will have length more than $2 \cdot \Delta$, where Δ is the visibility range of a spanner node and is provided as an input parameter to the algorithm. There will also be no edge in $E_{\mathtt{D}}$ with length more than δ, the maximum radius for neighborhoods in δ-PRM. Typically $\delta << \Delta$.

3.2 Algorithmic Concepts

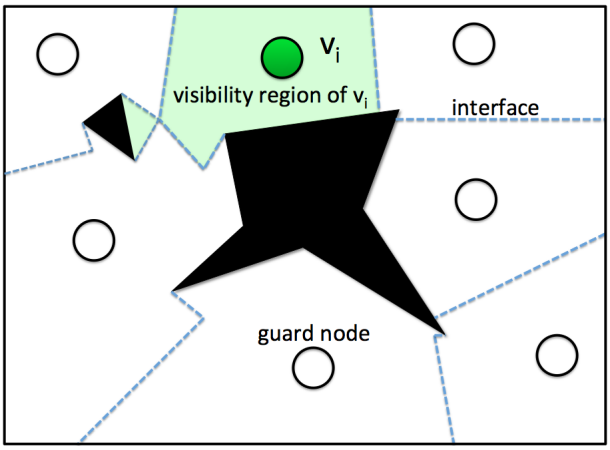

Figure 3.1: Visibility region of the representative v_i: the configurations that can be connected to v_i and have it as the closest node.

Representatives A spanner node $v \in V_{\mathtt{S}}$ will be the *representative* $rep(q)$ of sample $q \in V_{\mathtt{D}}$, if they can be connected with an obstacle-free path of length less than Δ and v is the closest to q node on the spanner with this property, i.e., $rep(q) = \{v \in V_{\mathtt{S}}$ so that $\mathtt{L}(v,q) \in \mathtt{C}_{free}$, $d(v,q) < \Delta$ and $\text{argmin}_{v \in V_{\mathtt{S}}}\ d(v,q)\}$. The *visibility region* of a spanner node v given the set of spanner nodes $V_{\mathtt{S}}$ is defined as the set of collision-free configurations that have v as their representative. Figure 3.1 offers an illustration of the visibility region notion.

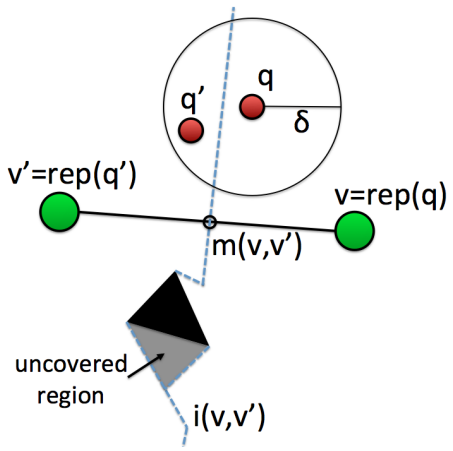

Figure 3.2: Two neighboring spanner nodes define an interface: the shared boundary of their visibility regions. q supports the interface $i(v, v')$, where v is the representative of q, if there is a configuration q' within the δ hyper-sphere of q that has a different representative ($rep(q') = v' \neq v$) and $\mathtt{L}(q, q') \in \mathtt{C}_{free}$.

Interfaces and support The *interface* between two spanner nodes v and v', $i(v, v')$, is the shared boundary of their visibility regions as in Figure 3.2. A sample $q \in V_{\mathtt{D}}$ *supports* the interface $i(v, v')$ of its representative $v = rep(q)$, if there is a sample $q' \in V_{\mathtt{D}}$ so that $\mathtt{L}(q, q') \in E_{\mathtt{D}}$ and $rep(q') = v'$, where $v' \neq v$.

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

$$m(v, v') \in \mathtt{L}(v, v') \text{ and } d(\ v,\ m(v, v')) = d(\ m(v, v'),\ v'\ )$$

If the local path $\mathtt{L}(v, v')$ is obstacle-free, then the midpoint $m(v, v')$ lies on the interface $i(v, v')$ between the two spanner nodes.

3.3 Algorithmic Details for Individual Steps

Algorithm 1 outlines the operation of `SPARS`. The input includes (i) the maximum number M of failures to add a node in `S`, (ii) t, the spanner's stretch factor, and (iii)

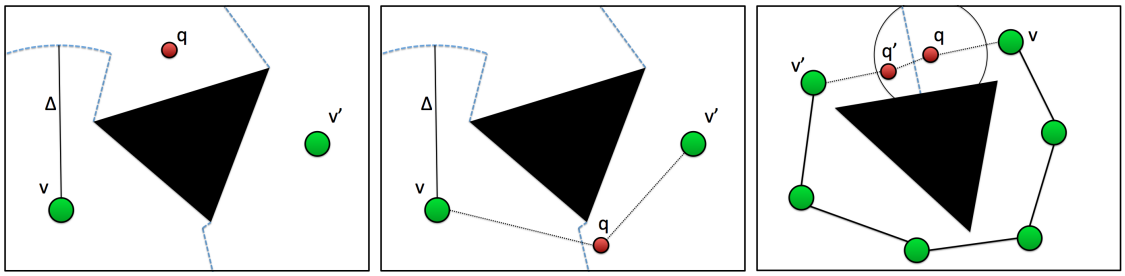

Figure 3.3: (left) q added as a guard. (middle) q added as a bridge. (right) q, q' support $i(v, v')$, which does not have an edge crossing it. Samples q and q' are candidates for addition.

the spanner visibility range Δ and (iv) the sample visibility range δ. The algorithm initializes the two graphs and sets the number of failed attempts to zero (line 1). While the failed attempts are fewer than M (line 2), the algorithm samples obstacle-free configurations q (line 3). Every q is added to the dense graph as a node and connected with an edge to existing samples $q' \in V_{\mathtt{D}}$ so that $d(q, q') < \delta$ and $\mathtt{L}(q, q') \in \mathtt{C}_{free}$ (line 4). Then, `SPARS` identifies the connections $E_q \in \mathtt{C}_{free}$ between the sample q and spanner nodes of distance less than Δ. Among these nodes, the representative $v = rep(q)$ is found (lines 5-11).

Figure 3.3 describes the first three cases under which `SPARS` adds a sample q to $V_{\mathtt{S}}$. If none of the existing spanner nodes can be connected to q, then the sample has to be added for coverage purposes (lines 12-13). The nodes added for `C`-space coverage will be called "guards". If q can connect two disconnected components, then it is also a candidate for addition (lines 14-15). The function `Add_Bridge_Spanner` first checks whether v_1 and v_2 can be connected directly with an edge. If they can, this edge is added, otherwise q is added and connected to v_1 and v_2. If a direct edge is added, its maximum length is $2 \cdot \Delta$. Note that the previous method `IRS2` employs this two ways of inserting samples into the spanner and also accepts nodes

if they introduce a "useful cycle", i.e., if they have two neighbors v_1 and v_2 so that $t \cdot (d(v_1, q) + d(q, v_2)) < \pi_{\mathtt{S}}(v_1, v2)$.

Beyond this point, the algorithm adds samples in $V_{\mathtt{S}}$ only if they support an interface. The algorithm detects if q has neighbors in `D` with a different representative, in which case, q lies on an interface (lines 17-19). If the representatives v and v' of q and q' do not share an edge, then q and q' are candidates for addition (lines 21-23). The objective is to guarantee that every two spanner nodes that share an interface will also share an edge in $E_{\mathtt{S}}$. This is handled by function `Add_Pair_Spanner` (Algorithm 2), which again first attempts to connect v and v' directly. If this is possible, the edge is added and no new node is introduced in `S`. If $\mathtt{L}(v, v')$ is not collision-free, then `Add_Pair_Spanner` will attempt to connect v and v' through $mid = m(q, q')$. If this is not possible, then both q and q' are added to `S`, and connected appropriately to their neighbors. `SPARS` proceeds to check whether q reveals paths that are much shorter than spanner paths (lines 24-35).

For all interfaces $i(v, v')$ that q supports (line 25), the algorithm considers neighbors v'' of v, so that v'' is not linked to v' (line 26). Figure 3.4 gives an example. For such a case, the algorithm computes the spanner path $\pi_{\mathtt{S}}(m(v, v'), m(v, v''))$ (line 27) and similar spanner paths between v and neighbors x of v that do not share an edge with v' (lines 28-29). Among all of them, the algorithm stores the longest path (line 30). The algorithm aims to guarantee that all these spanner paths are less than t times longer than the

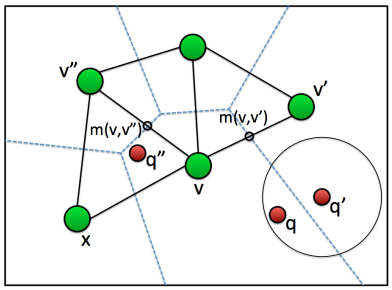

Figure 3.4: q, q' support $i(v, v')$, while q'' supports $i(v, v'')$. If path $\pi_{\mathtt{D}}(q, q'')$ is shorter than t times the length of spanner paths, then samples along $\pi_{\mathtt{D}}(q, q'')$ are candidates for addition.

shortest paths between samples along $i(v, v')$ and $i(v, v'')$ (lines 31-35).

The algorithm computes the shortest path between q, which supports $i(v, v')$, and all samples that support $i(v, v'')$ and have v as their representative (lines 31-35). If the shortest path $\pi_{\mathtt{D}}(q, q'')$ in `D` is shorter than t times the length of the longest corresponding spanner path then the algorithm considers the samples along $\pi_{\mathtt{D}}(q, q'')$ for addition (lines 34-35). This is accomplished by function `Add_Path_Spanner` (Algorithm 3). If q has not been added to the spanner, the parameter $failures$ is increased (lines 36-37). An efficient implementation of `Add_Path_Spanner` would first try to directly connect v with v'', and it would also try to smooth the final path added to `S`.

The method `Reduce_Interfaces` is called by all functions that add nodes to $V_{\mathtt{S}}$. It aims to directly connect vertices which share an interface but not an edge using the local planner. Doing so prevents the algorithm from needing to add samples q and q' that support this interface.

Algorithm 1: SPARS($\mathtt{C}_{free}, M, t, \Delta, \delta$)

$failures \leftarrow 0$; $V_D \leftarrow \emptyset$; $E_D \leftarrow \emptyset$; $V_S \leftarrow \emptyset$; $E_S \leftarrow \emptyset$;
while *failures* $< M$ **do**
  $q \leftarrow$ Sample($\mathtt{C}_{free}$);
  Add_Node_δ-PRM(D, q, δ);
  $E_q = \emptyset$; // initialize edges from q to spanner nodes to empty
  $v = NULL$; // v will be q's representative
  for $v' \in V_S$ **do** // Find guards that can be connected to q
    if $d(v', q) < \Delta$ *and* $\mathtt{L}(v', q) \subset \mathtt{C}_{free}$ **then**
      $E_q \leftarrow E_q \cup \{\mathtt{L}(v', q)\}$; // add local path
      if $v == \emptyset$ *or* $d(v', q) < d(v, q)$ **then**
        $v = v'$;
  if $E_q == \emptyset$ **then** // when q not visible from existing guards
    Add_Guard_Spanner(V_S, E_S, q);
  else if q *connects two spanner nodes* v_1, v_2 *that were previously disconnected* **then**
    Add_Bridge_Spanner(V_S, E_S, v_1, v_2, q);
  else
    $V' = \emptyset$; // V': nodes with interface supported by q
    for *all* $q' \in V_D$ *so that* $\mathtt{L}(q, q') \in E_D$ **do**
      if $rep(q') \neq rep(q)$ **then** $V' \leftarrow V' \cup rep(q')$
    if $V'! = \emptyset$ **then**
      for $v' \in V'$ **do**
        if $\mathtt{L}(v, v') \notin E_S$ **then**
          Add_Pair_Spanner(V_S, E_S, v, v', q, q')
      if $q \notin V_S$ **then**
        for $v' \in V'$ **do**
          for *all* $v'' \in V_S, v'' \neq v'$ *so that* $\mathtt{L}(v, v'') \in E_s$, $\mathtt{L}(v', v'') \notin E_S$ **do**
            $\Pi_{\mathtt{S}} \leftarrow \{\pi_{\mathtt{S}}(m(v, v'), m(v, v''))\}$; // midpoint paths
            for *all* $x \in V_S, x \neq v, v', v''$: $\mathtt{L}(x, v), \mathtt{L}(x, v'') \in E_s$, $\mathtt{L}(x, v') \notin E_S$ **do**
              $\Pi_{\mathtt{S}} \leftarrow \Pi_{\mathtt{S}} \cup \{\pi_{\mathtt{S}}(m(v, v'), m(v, x))\}$;
            $\pi_{\mathtt{S}} = argmax_{\forall \pi \in \Pi_{\mathtt{S}}} |\pi|$; // max. length path
            while $q \notin V_S$ **do**
              for *all* q'' *that support* $i(v, v'')$ **do**
                $\pi_{\mathtt{D}} \leftarrow$ Shortest_Path(D, q, q'');
                if $t \cdot |\pi_{\mathtt{D}}| < |\pi_{\mathtt{S}}|$ **then**
                  Add_Path_Spanner($V_S, E_S, V_D, E_D, \pi_{\mathtt{D}}, v, v''$);
    if $q \notin V_S$ **then** $failures++$;
    else $failures = 0$;
return (V_S, E_S);

Algorithm 2: `Add_Pair_Spanner` (S, D, v, v', q, q')

$mid \leftarrow m(q, q')$;
if $\mathtt{L}(v, v') \in \mathtt{C}_{free}$ **then**
    $E_\mathtt{S} \leftarrow E_\mathtt{S} \cup \{\mathtt{L}(v, v')\}$;
else if $\mathtt{L}(v, mid) \in \mathtt{C}_{free}$ *and* $\mathtt{L}(v', mid) \in \mathtt{C}_{free}$ **then**
    $V_\mathtt{S} \leftarrow V_\mathtt{S} \cup \{mid\}$;
    $E_\mathtt{S} \leftarrow E_\mathtt{S} \cup \{\mathtt{L}(v, mid), \mathtt{L}(v', mid)\}$;
    `Reduce_Interfaces`(S, D, mid);
else
    $V_\mathtt{S} \leftarrow V_\mathtt{S} \cup \{q, q'\}$;
    $E_\mathtt{S} \leftarrow E_\mathtt{S} \cup \{\mathtt{L}(v, q), \mathtt{L}(q, q'), \mathtt{L}(q', v')\}$;
    `Reduce_Interfaces`(S, D, q);
    `Reduce_Interfaces`(S, D, q');

Algorithm 3: `Add_Path_Spanner` $(V_\mathtt{S}, E_\mathtt{S}, V_\mathtt{D}, E_\mathtt{D}, \pi, v, v'')$

$prior \leftarrow v$;
for $q \in \pi$ **do**
    $V_S \leftarrow V_S \cup \{q\}$;
    $E_S \leftarrow E_S \cup \{\mathtt{L}(prior, q)\}$;
    $prior \leftarrow q$;
$E_S \leftarrow E_S \cup \{\mathtt{L}(prior, v'')\}$;
for $q \in \pi$ **do**
    `Reduce_Interfaces` (S, D, q);

Algorithm 4: `Reduce_Interfaces` (S, D, v)

for $v' \in V_S$ *s.t.* $d(v, v') < 2 \cdot \Delta$ **do**
    if $\mathtt{L}(v, v') \notin E_S$ **then**
        for $(q, q') \in V_D$ **do**
            if $d(q, m(v, v')) < \delta$ *and* $d(q', m(v, v')) < \delta$ **then**
                if $rep(q) \neq rep(q')$ **then**
                    if $\mathtt{L}(v, rep(q')) \in \mathtt{C}_{free}$ **then**
                        $E_\mathtt{S} \leftarrow E_\mathtt{S} \cup \{\mathtt{L}(v, rep(q'))\}$;

Chapter 4 Attained Properties of SPARS

This chapter shows the properties of the resulting Roadmap Spanner, S. It begins by stating explicitly what assumptions and requirements are in place in order for these properties to exist. Proofs of the correctness of these properties are also provided.

4.1 Requirements and Assumptions

The existence of a finite set of nodes that can even achieve just coverage depends on properties of $\mathtt{C}_{free}$. Such a finite set may not always exist, such as in a fractal space. This work assumes that the C-space is expansive, similar to related literature [5, 6].

Lookout *The lookout of a set Q is the subset of configurations q in Q, such that the measure of what is visible from q outside of Q is greater than some fraction of the measure of the part of the free space outside of Q. More formally:*

$$\mathtt{LOOKOUT}_{\beta}(Q) = \{q \in Q \mid \mu(V(Q) \setminus Q) \geq \beta\mu(\mathtt{C}'_{free} \setminus Q)\}$$

where $\mathtt{C}'_{free}$ is a $\mathtt{C}_{free}$ connected component and $V(Q)$ is the visibility of set Q.

ϵ, α, β-expansiveness *A space is expansive if it satisfies:*

- $\forall\ q \in \mathtt{C}'_{free}$, where $\mu(V(q)) \geq \epsilon\mu(\mathtt{C}_{free})$
- $\forall\ Q \subseteq \mathtt{C}'_{free}$, where $\mu(\mathtt{LOOKOUT}_{\beta}(Q)) \geq \alpha\mu(Q)$

The author assumes SPARS operates in an expansive space, and all theorems and lemmas rely on the space being expansive, as even coverage cannot be provided by

sampling-based methods in poorly behaved environments, such as fractal spaces.

4.2 Probabilistic Completeness

In general, sampling-based methods are unable to provide guarantees on completeness, and instead must provide a weaker guarantee called probabilistic completeness. Probabilistic completeness states that if a solution to a given problem instance exists, then an algorithm with this property will return that solution with probability one as computation tends to infinity. If, however, no solution is returned by the algorithm, it is unknown whether there does not exist a solution to the problem, or if the algorithm has simply been given insufficient time to find the solution. In order to show that an algorithm has probabilistic completeness as a property, it is sufficient to show that it has the properties of *coverage* and *connectivity*.

4.2.1 Providing Coverage

To provide coverage, it has to be that the resulting spanner allows every collision-free configuration to be connected with a spanner node through a collision-free path.

Theorem 4.2.1 (Coverage) *Upon termination of* SPARS *and with probability* $1-\frac{1}{M}$*, the following is true,* $\forall\ q\ \in \mathtt{C}_{free} : \exists\ v \in V_{\mathtt{S}}$ *so that* $\mathtt{L}(q, v) \in \mathtt{C}_{free}$.

The detailed argument to support the above statement can be found in the work that presented the Visibility PRM [23] as SPARS is following a similar approach regarding C-space coverage and the stopping criterion. It also relates to the property of ϵ-goodness that expansive spaces provide [6]. At a high level, each new guard inserted

to S increases the coverage of $\mathtt{C}_{free}$ and the probability of generating configurations in non-covered regions decreases over iterations. 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. This is a conservative estimate, since the algorithm assumes an artificial visibility range limit Δ for spanner nodes. The work on Visibility PRMs has shown that even for relatively 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.

4.2.2 Providing Connectivity

Theorem 4.2.2 (Connectivity) *Upon termination of* SPARS *and with probability* 1 *as M goes to infinity, for every pair of nodes $v, v' \in V_{\mathtt{S}}$ that are connected with a collision-free path in $\mathtt{C}_{free}$, there is a path $\pi_{\mathtt{S}}(v, v')$ that connects them on* S.

The above property arises from lines 14-15 of SPARS. 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, since every configuration will be eventually sampled. The combination of the last two theorems provides probabilistic completeness, at least when the algorithm samples configurations q in a uniform way.

4.3 Proving Asymptotic Optimality of D

An important property of the algorithm is that D has asymptotic optimality as a property. This is because the sparse roadmap, S, is not a spanner of the implicity graph G, but rather of D. Thus, we must show that D converges to G as computation tends to infinity. By using δ-PRM, the approach is able to construct a dense graph that provides asymptotic optimality [9]. The use of δ-PRM is also in part due to the algorithm, as δ is used as an input parameter of the algorithm. Thus, the following lemma holds.

Lemma 4.3.1 (Asymptotic Optimality of D) *Upon termination of* SPARS *and with probability* 1 *as* $M \to \infty$*, for every cl−robust optimal path* $\pi_{\mathtt{C}}(q, q')$*, there is a collision-free path on the dense graph* $\pi_{\mathtt{D}}(q, q')$*, so that:* $|\pi_{\mathtt{D}}(q, q')| \to |\pi_{\mathtt{C}}(q, q')|$.

4.4 Showing the Existence of Asymptotically Near-Optimal Paths in S

If the length of spanner paths is bounded by the dense paths, they will be asymptotically bounded relative to optimal C-space paths, since D converges to optimal. Now, it must be shown that paths through S exist and have the desired properties.

Lemma 4.4.1 (Coverage of Optimal Paths by S) *Consider the shortest dense path* $\pi_{\mathtt{D}}(q_0, q_m)$*. As* M *goes to infinity, the probability of having a sequence of guard nodes* $U = (v_1, \ldots, v_n)$ *with the following properties goes to 1:*

- *Each* q *along* $\pi_{\mathtt{D}}(q_0, q_m)$ *belongs to the visibility region of a spanner node.*
- v_1 *is the representative of* q_0*,* $v_1 = rep(q_0)$ *and similarly* $v_n = rep(q_m)$.

- *All the paths* $\mathtt{L}(v_i, v_{i+1})$ *belong to the set* E_S *of the sparse roadmap spanner.*

Proof Fig. 4.1 (left) illustrates the first property, which is a direct outcome of the coverage theorem. The second point holds without loss of generality. The following discussion focuses on the validity of the last point and relates to Fig. 4.1 (middle). Consider q_j along $\pi_{\mathtt{D}}(q_0, q_m)$ that supports $i(v_i, v_{i+1})$. With probability 1 as $M \to \infty$, samples have been generated in the $\frac{cl}{2}$ neighborhood of q_j, which is guaranteed to be collision-free. Consider two samples in the $\frac{cl}{2}$-ball centered at q_j: (a) q_j^i with spanner node v_i as its representative and (b) q_j^{i+1} with v_{i+1} as its representative.

To guarantee the algorithm will produce such samples, it has to be that the set of such configurations has non-zero measure. The region from which the sample q_j^i must be generated from is the intersection of (a) the volume of visibility paths from v_i to the optimum path $\pi_{\mathtt{D}}$ and (b) the $\frac{cl}{2}$-ball centered at q_j, which has positive measure given expansiveness [6]. An expansive space can be partitioned so that the $\frac{cl}{2}$-ball is one partition of the space. Expansiveness states that every subset of both partitions must have visibility of the other with positive measure. It is already true that the local path $\mathtt{L}(v_i, q_j)$ is collision-free and contributes towards the lookout of the partition containing v_i. From the above observations, a sample q_j^i will be generated and similarly for q_j^{i+1}.

The local path $\mathtt{L}(q_j^i, q_j^{i+1})$ must be collision-free. Furthermore, for $\delta > 0$ there are q_j^i and q_j^{i+1} for which $d(q_j^i, q_j^{i+1}) < \delta$. Then, the algorithm would have added the edge $\mathtt{L}(q_i^i, q_i^{i+1})$ in $E_{\mathtt{D}}$. There are two cases at this point. Either v_i and v_{i+1} are connected as desired, or the representatives are not connected. Assume v_i and v_{i+1} are not connected. Then all the requirements of the lines 17-23 of the algorithm

are satisfied: (i) $q_j^i, q_j^{i+1} \in V_D$ and $\mathtt{L}(q_j^i, q_j^{i+1}) \in E_{\mathtt{D}}$, (ii) $rep(q_j^i) = v_i \neq v_{i+1} = rep(q_j^{i+1})$ (iii) $\mathtt{L}(v_i, v_{i+1}) \notin E_S$. Thus, `Add_Pair_Spanner` would have been called. The function would either introduce the edge $\mathtt{L}(v_i, v_{i+1})$ or it would add the samples q_j^i, q_j^{i+1} to S together with the edges $\mathtt{L}(v_i, q_j^i), \mathtt{L}(q_j^i, q_j^{i+1}), \mathtt{L}(q_j^{i+1}, v_{i+1})$. This means that $(v_1, \ldots, v_i, v_{i+1}, \ldots, v_n)$ can be replaced by $(v_1, \ldots, v_i, q_j^i, q_j^{i+1}, v_{i+1}, \ldots, v_n)$, which still covers the dense path $\pi_{\mathtt{D}}(q_0, q_m)$. In this case, all the nodes between v_i and v_{i+1} are pairwise connected. Thus, if an edge doesn't exist, a new sequence can always be created where all the guards are pairwise connected. ∎

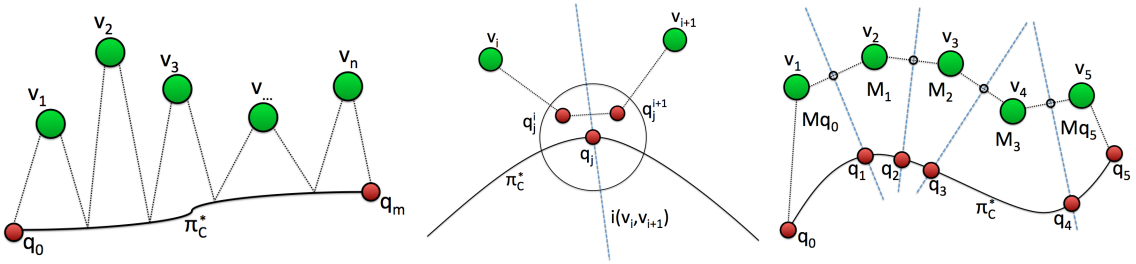

Figure 4.1: (left) For every $\pi_{\mathtt{D}}(q_0, q_m)$, there is a sequence of $V_{\mathtt{S}}$ nodes $\{v_1, \ldots, v_n\}$ so that every q along the path is "visible" by at least one v_i. For each (v_i, v_{i+1}) there is an edge $\mathtt{L}(v_i, v_{i+1})$ in $E_{\mathtt{S}}$. (middle) If there is no edge $\mathtt{L}(v_i, v_{i+1})$, the samples q_i^i and q_i^{i+1} would have been added to $\mathtt{S}$. (right) The spanner path $\pi_{\mathtt{S}}(q_0, q_5)$ can be split into: $\{M_{q_0}, M_1, M_2, M_3, M_{q_5}\}$. Each M_i is a path of the form $\pi_{\mathtt{S}}(m(v_i, v_{i+1}), m(v_{i+1}, v_{i+2}))\}$. The first and last segments connect q_0 and q_5 to the spanner path.

A solution path $\pi_{\mathtt{S}}(q_0, q_m)$ computed through the spanner will correspond to the concatenation of $\mathtt{L}(q_0, v_1)$ and $\mathtt{L}(v_n, q_m)$ with local paths $\mathtt{L}(v_i, v_j)$ along the sequence $U = (v_1, \ldots, v_n)$. Note that not all consecutive guard edges of the type $\mathtt{L}(v_i, v_{i+1})$ need to be part of the shortest spanner path, as there can be shortcuts along U. Then, the path on the spanner can be decomposed into segments $M_i = \pi_{\mathtt{S}}(m(v_{i-1}, v_i), m(v_i, v_{i+1})))$, which start at a midpoint between two spanner nodes and stop at another midpoint. There are also the following special segments: $M_{q_0} = (\mathtt{L}(q_0, v_1), \mathtt{L}(v_1, m(v_1, v_2)))$ and $M_{q_m} = (\mathtt{L}(m(v_{n-1}, v_n), v_n), \mathtt{L}(v_n, q_m))$. These

segments connect the start or the final configuration with spanner midpoints. Figure 4.1 (right) provides an illustration. Overall: $\pi_S(q_0, q_m) \equiv (M_{q_0}, M_1, \ldots, M_k, M_{q_m})$.

Lemma 4.4.2 (Connection Cost) *The length of M_{q_0} and M_{q_m} of a path between configurations q_0, q_m computed through the spanner is bounded by:* 4Δ.

Proof Consider the segment M_{q_0}. The length of $\mathtt{L}(q_0, v_1)$ cannot be longer than Δ because v_1 is the representative of q_0. Similarly, the distance between v_1 and $m(v_1, v_2)$ cannot be more than Δ, since $m(v_1, v_2)$ is the midpoint between two spanner nodes that share an edge. Overall $|M_{q_0}| < \frac{4 \cdot \Delta}{2}$. The same bound can be shown for the segment M_{q_m}. Consequently: $|M_{q_0}| + |M_{q_m}| < 4 \cdot \Delta$. ∎

Lemma 4.4.3 (Spanner Property) *Segments M_i have length bounded by $t \cdot |\pi_{\mathtt{D}}(q_{i-1}, q_i)|$, where q_i lies at the intersection of $\pi_{\mathtt{D}}(q_0, q_m)$ with the interface $i(v_i, v_{i+1})$.*

Proof The arguments presented here correspond to Fig. 4.2. Given lemma 4.4.1, the edges $\mathtt{L}(v_{i-1}, v_i)$ and $\mathtt{L}(v_i, v_{i+1})$ must belong to the set $E_{\mathtt{S}}$, as the algorithm ensures that all interfaces will eventually have edges connecting the vertices inducing them. Assume the true optimal path goes through the visibility region of vertex v_i on sequence U. There are two cases to be considered: (a) there is no edge $\mathtt{L}(v_{i-1}, v_{i+1})$ in E_S (Fig. 4.2 (left)) or (b) there is such an edge (Fig. 4.2 (right)).

To ensure that the spanner property is upheld in the first case, it is sufficient that the path on $\mathtt{S}$ from $m(v_{i-1}, v_i)$ to $m(v_i, v_{i+1})$ is shorter than t times the length of the shortest dense paths between interfaces $i(v_{i-1}, v_i)$ and $i(v_i, v_{i+1})$. Lines 24-35 of `SPARS`, however, ensure that this property is true. When the newly sampled

configuration q, is on an interface $i(v, v')$, the algorithm will search for paths from v' to all v'' where $\mathtt{L}(v', v'') \notin E_{\mathtt{S}}$. The equivalent of v in Fig. 4.2 (left) is v_i, v' would be v_{i+1} and v'' would be v_{i-1} or vice verca.

In the second case, the algorithm will not check the interface $i(v_{i-1}, v_i)$ and $i(v_i, v_{i+1})$, because the nodes v_{i-1} and v_{i+1} share an edge. Nevertheless, the algorithm considers additional spanner paths when it compares against the length of dense paths. In particular, the algorithm will check whether the spanner path from $m(v_{i-2}, v_{i-1})$ to $m(v_{i-1}, v_{i+1})$ is shorter than t times the length of the shortest dense paths between the interfaces $i(v_{i-2}, v_{i-1})$ and $i(v_{i-1}, v_i)$. Notice the difference in the second vertex of the second midpoint and the second interface. In this case, node v_{i-1} in Fig. 4.2 (right) is equivalent to node v in lines 24-35 of `SPARS`, v_{i-2} is equivalent to node v', v_i is equivalent to v'' and v_{i+1} is equivalent to node x. The algorithm will also provide the same guarantee for the spanner path $m(v_{i-1}, v_{i+1})$ to $m(v_{i+1}, v_{i+2})$ and overall: $|\pi_{\mathtt{S}}(m(v_{i-2}, v_{i-1}), m(v_{i+1}, v_{i+2}))| < t \cdot (|\pi_{\mathtt{D}}(q_{i-2}, q_{i-1})| + |\pi_{\mathtt{D}}(q_i, q_{i+1})| < t \cdot (|\pi_{\mathtt{D}}(q_{i-2}, q_{i-1})| + |\pi_{\mathtt{D}}(q_{i-1}, q_i)| + |\pi_{\mathtt{D}}(q_i, q_{i+1})|)$ and the spanner property is still satisfied. If the spanner property is violated, `SPARS` will call function `Add_Path_Spanner`, which will either directly connect the corresponding vertices or add the dense path into the spanner graph. ∎

Theorem 4.4.4 (Asymptotic Near-Optimality with Additive Cost) *Upon termination of* `SPARS`*, as* $M \to \infty$ *and* $\forall q_0, q_m \in \mathtt{C}_{free}$*:* $|\pi_{\mathtt{S}}(q_0, q_m)| < t \cdot |\pi_{\mathtt{D}}(q_0, q_m)| + 4 \cdot \Delta$.

Proof Lemma 4.4.1 provides the existence of a sequence U of spanner nodes that can be used to solve path planning queries for any q_0, q_m (i.e., it is possible to connect q_0 and q_m to the sequence U, and all the nodes in the sequence are pairwise connected).

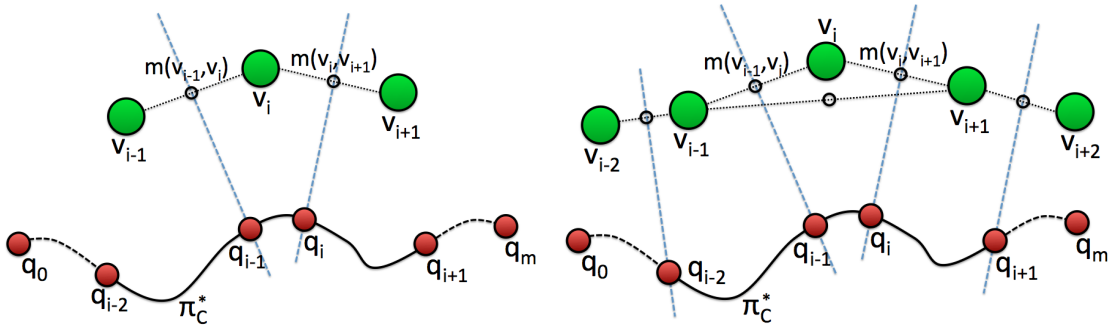

Figure 4.2: (left) The optimal path between q_{i-1} and q_i is represented by the segment M_i on the spanner. The algorithm checks whether the length of M_i is less than $t \cdot |\pi_{\mathtt{D}}(q_{i-1}, q_i)|$. (right) Nodes u_{i-1} and u_{i+1} are connected with an edge in this case. Thus, the spanner path will not contain the node u_i. Nevertheless, the algorithm checks whether the spanner path $[m(u_{i-2}, u_{i-1}), u_{i-1}], [u_{i-1}, m(u_{i-1}, u_{i+1})]$ is shorter than t times the shortest dense path between the interfaces $i(u_{i-2}, u_{i-1})$ and $i(u_{i-1}, u_{i+1})$. The same is true for $i(u_{i-1}, u_{i+1})$ and $i(u_{i+1}, u_{i+2})$.

The combination of lemmas 4.4.2 and 4.4.3 indicates that the solution path $\pi_{\mathtt{S}}(q_0, q_m)$ computed through this sequence has length that is bounded relative to the length of the dense path $|\pi_{\mathtt{S}}(q_0, q_m)| < t \cdot |\pi_{\mathtt{D}}(q_0, q_m)| + 4 \cdot \Delta$, where the additive cost arrises from the cost of connecting the query points to the spanner nodes. Lemma 4.3.1 indicates that the paths computed through the dense graph converge asymptotically to the optimal ones $\pi_{\mathtt{S}}(q_0, q_n)$. The combination of all these lemmas proves this theorem. ∎

4.5 On the Convergence to a Finite Data Structure

An important issue is whether `SPARS` adds an infinite number of nodes to `S` in order to provide the prior path quality properties. Should `SPARS` be capable of doing so, it will show an interesting result: that the paths through a continuous space can be approximated within a bound by a finite data structure. As a first step toward this, it is shown that the algorithm ceases to add nodes to `S` as computation tends to

infinity.

Theorem 4.5.1 (Termination of Node Addition) *As the number of iterations goes to infinity, the probability of* SPARS *adding nodes to* S *goes to zero.*

Proof There are four ways with which SPARS may add nodes in the spanner S and is necessary to show that the probability of adding each type of node goes to zero.

Nodes for coverage are added when a sample q lies outside the visibility range of all existing nodes in $V_{\mathtt{S}}$. Theorem 4.2.1 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 q connects two disconnected components. Eventually, enough nodes for ensuring coverage will be added in S. Given the number of nodes n at that point, the number of samples that can connect disconnected components is finite.

Nodes for ensuring interfaces have edges are added when a sample q has a neighbor q' in D and their representatives v and v' do not share an edge. Assuming coverage and connectivity has been ensured, it is possible there are interfaces without an edge. In the general case, when a node is added to S, multiple new interfaces are defined. But the algorithm employs the function Reduce_Interfaces, which attempts to connect a new node to all nodes with which it shares an interface. There are three cases for two neighboring nodes: (i) The interface lies outside $\mathtt{C}_{free}$ and there is no need to add an edge. (ii) $\mathtt{L}(v, v')$ lies within $\mathtt{C}_{free}$ and the edge can be immediately added. (iii) $\mathtt{L}(v, v')$ is not within $\mathtt{C}_{free}$, but there exists a portion of the interface in $\mathtt{C}_{free}$. In this case, new nodes need to be added, which, however, will also add new

interfaces, which need to be connected by an edge. Nevertheless, the newly generated interfaces will eventually have to fall into case (i) or case (ii) if the C-space is nicely behaved. Otherwise, it has to be that every time a node is added, there will exist an obstacle which partially covers the local path to neighbors with which the node shares an interface. Overall, all interfaces will be eventually connected with an edge, so the probability of adding nodes for this reason goes to 0.

Nodes for ensuring path quality are added when a sample q supports an interface and reveals a path in D that is significantly shorter than the corresponding path in S. SPARS always adds the minimal amount of nodes possible while ensuring that the path quality properties of the algorithm hold. When adding a dense path to S, SPARS will first try to smooth this path. In some cases, the path cannot be smoothed, which will result in the addition of nodes to S. The addition of new nodes could potentially always create new visibility regions where a dense path could be found that is sufficiently shorter than the spanner paths. Eventually, however, the spanner nodes which the algorithm attempts to connect will be within cl of each other and it would be guaranteed that $\mathtt{L}(v, v')$ is collision-free and the nodes can be connected directly. Therefore, SPARS will eventually stop adding nodes in this fashion.

Chapter 5 Algorithm Implementation and Experimental Results

The algorithm was implemented in a simulation software written in C++. A concern for the implementation of the method was its space requirements. To improve space efficiency, information was kept to a minimum on the dense graph's nodes and edges. Dense nodes take up approximately 40 bytes of memory in $SE(2)$ examples and 72 bytes in $SE(3)$ examples. Sparse nodes use a larger amount of memory, as they retain a lists of all the nodes in the dense graph which they represent, introducing a cost of 4 bytes per dense node. All edges use approximately 12 bytes of memory.

The implementation performed several optimizations compared to the algorithm described. In order to speed up convergence, connections are attempted to neighbors of newly added sparse nodes in the spanner. The neighbors considered are all which are within $2 * \Delta$ distance. It often happens that the algorithm will not be able to quickly identify that two spanner nodes share an interface, as not enough samples will be present in the dense graph `D` to correctly identify the interfaces. Furthermore, whenever paths are added to the spanner, the algorithm always attempts to directly connect points, and if it cannot, it will try smoothing the paths.

For nearest neighbors queries, kd-trees were used for fast query resolution. Nevertheless, it is often much more practical to consider the neighbors of nodes already in the spanner or the graph and then prune them according to their distance to the selected node. This technique was used heavily, especially for querying the dense graph.

In order to validate the theoretical bounds given, experimental results on the algorithm are provided. The results support the theoretical guarantees and show examples of the algorithm's performance.

5.1 Experimental Set-up

Experiments were run in various environments in $SE(2)$ and $SE(3)$. Four environments were tested, as shown in Figure 5.1 for kinematic rigid bodies. The 2D Maze used a disk system, while the Teeth environment used a 'Z' shaped body. For both cases in $SE(3)$, a variation of the 'Z' robot was used which has one of its legs coming off in an orthogonal direction to the other leg. Runs in $SE(2)$ were tested with $M = 4000$, $t = 3$, and $\Delta = 20$, while runs in $SE(3)$ were tested with $M = 1400$, $t = 3$, and $\Delta = 25$. All runs were terminated if the stopping criterion was not satisfied after 60 minutes `IRS2` experiments were run instead with a 15 minute time-limit.

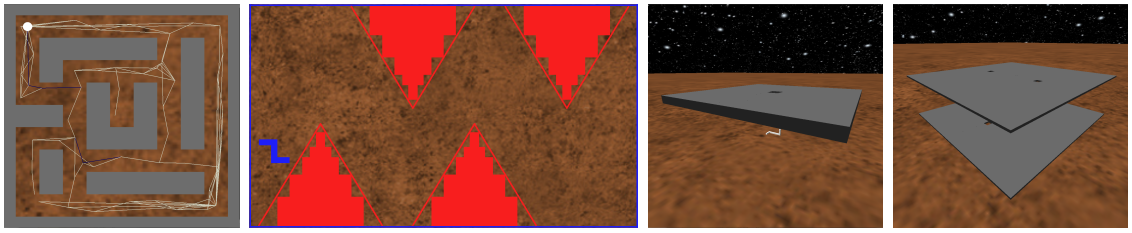

Figure 5.1: Environments for testing from left to right: 2D Maze ($SE(2)$), Teeth ($SE(2)$), 3D Hole ($SE(3)$), and Many Holes ($SE(3)$). The 2D Maze tests performance for constraining spaces, while the Teeth environment tests algorithmic performance in large, open spaces.

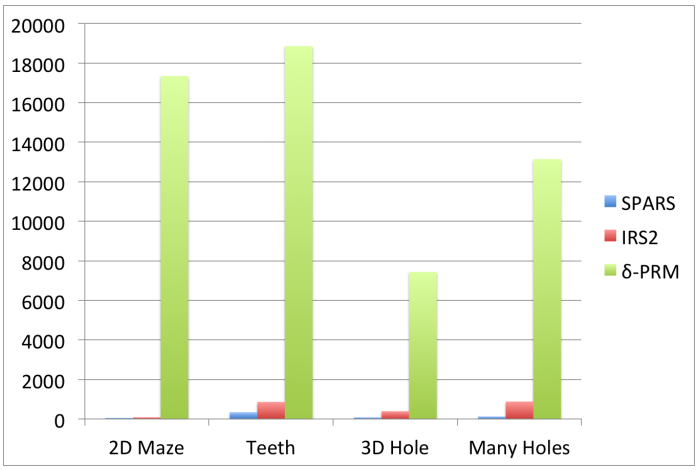

Figure 5.2: A comparisson of the number of nodes in the planning structures for the various algorithms. Notice SPARS and IRS2 have much fewer nodes than PRM*. It is also expected that SPARS would have fewer nodes than IRS2, as it aims to always prevent the addition of nodes to the roadmap.

5.2 Results and Comparison

SPARS was compared against asymptotically optimal and near-optimal planners, in particular: δ-PRM and IRS2. The algorithms were compared in terms of path quality, query resolution time, and number of nodes and edges in the final roadmaps. Results shown for δ-PRM were extracted from the dense graph D constructed by the SPARS algorithm. The goal of SPARS is to reduce the number of nodes in the graph, and Figure 5.2 shows the reduction of nodes provided by SPARS. Note how the algorithm returns orders of magnitude less nodes than PRM*. This indicates that the properties of the space are being captured with only a very small subset of nodes. Furthermore, the overall number of edges in SPARS is also orders of magnitude less than PRM*, as seen in Figure 5.3. SPARS still adds a large number of connecting edges between nodes, but because there are so few nodes, the resulting size of the graph is very small. Due to the reduced size of the roadmap, query resolution times are also much shorter than

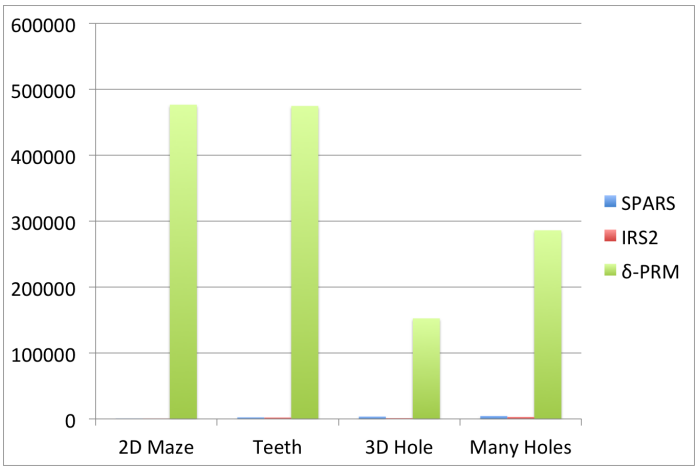

Figure 5.3: A comparisson of the number of edges in the resulting structures. Though both SPARS and IRS2 are orders of magnitude smaller than PRM*, the number of edges in SPARS is larger than in IRS2.

PRM*, as seen in Figure 5.4. An important issue is path quality, as the results should validate that paths are within the theoretical bounds drawn. Figure 5.5 shows that not only are path lengths within these bounds, but are typically only 110-120% of the length of paths returned by PRM*. This shows that even given bounds for a stretch factor of 3, which would give bounds of at least 300% of PRM*, the paths returned by SPARS are much lower than this theoretical bound.

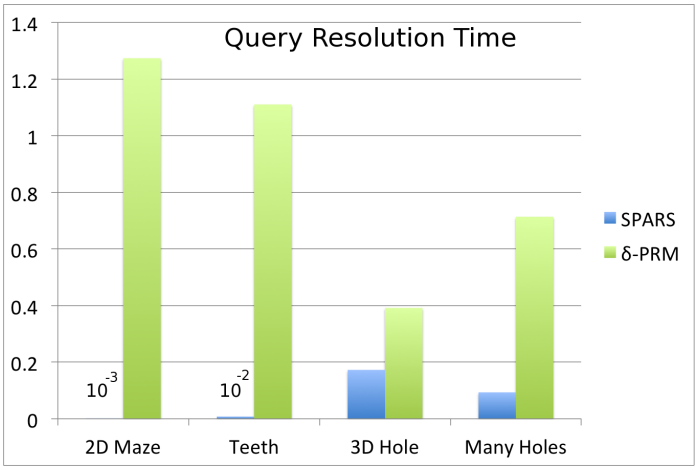

Figure 5.4: Query resolution time of SPARS when compared to PRM*. The decrease in query resolution time is due to the much smaller planning structure provided by SPARS.

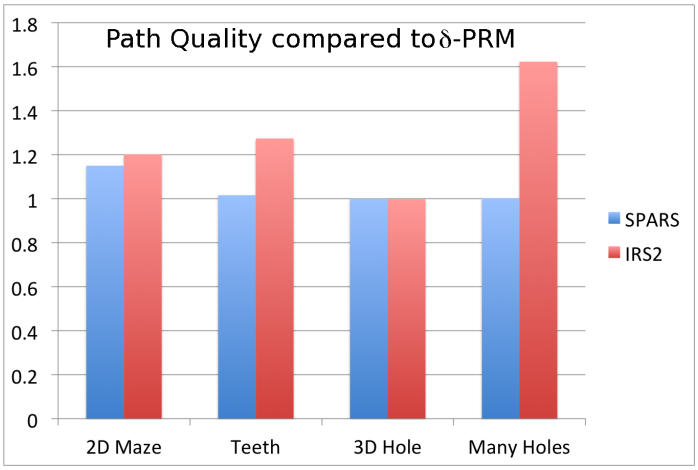

Figure 5.5: Though theoretical bounds on path quality are fairly loose, the actual path lengths returned by SPARS are only 110%-120% of the path lengths returned by PRM*.

Chapter 6 Future Work and Directions

This thesis describes an approach for computing a sparse roadmap structure for answering path queries in continuous spaces while maintaining near-optimality guarantees. The author is hopeful that, because nodes will eventually stop being added to the roadmap, SPARS provides a finite-sized data structure. Simulations indicate that the resulting graph is many of orders sparser (i.e., it has fewer nodes and edges) than graphs with asymptotic optimality guarantees. This results in significantly shorter query resolution times. The quality of paths computed on the sparse roadmap spanner is shown to be in practice significantly better than the theoretical guarantees.

The most important direction for this research is figuring out whether S converges to a finite-sized structure, or if an infinite number of nodes are still needed. Future research will address the dependency of the current approach to the input parameters, such as the visibility ranges Δ, δ and the maximum allowed number of failures M. An interesting question is whether the proposed method can discover the important homotopic classes in a C-space and how does it compare against methods that aim to identify homotopic classes directly [7, 22]. It is also important to consider alternative methods that do not need to store during the construction of the spanner the entire graph returned by the asymptotically optimal planner. Such a development could reduce the space and time requirements of the spanner's construction. In a similar direction, it will be helpful to introduce steps that improve computational efficiency, such as tools for reducing the cost of A* searches on the dense graph or the cost of nearest-neighbor queries.

Bibliography

[1] P. Agarwal. Compact Representations for Shortest-Path Queries. Appeared at the IROS 2012 Workshop on Progress and Open Problems in Motion Planning, September 2011.

[2] 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.

[3] Surender Baswana and Sandeep Sen. A Simple and Linear Time Randomized Algorithm for Computing Spanners in Weighted Graphs. *Random Structures and Algorithms*, 30(4):532–563, July 2007.

[4] 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.

[5] 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.

[6] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized Kinodynamic Motion Planning with Moving Obstacles. *IJRR*, 21(3):233–255, March 2002.

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

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

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

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

[11] 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.

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

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

[14] 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.

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

[16] James D Marble and Kostas E Bekris. Asymptotically Near-Optimal is Good Enough for Motion Planning. In *ISRR*, Flagstaff, AZ, August 2011.

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

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

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

[20] 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.

[21] G. Sanchez and J.-C. Latombe. A Single-Query, Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking. In *ISRR*, pages 403–418, 2001.

[22] 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.

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

[24] G. Varadhan and D. Manocha. Star-shaped Roadmaps: A Deterministic Sampling Approach for Complete Motion Planning. *IJRR*, 2007.

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