APPROXIMATING CONFIGURATION SPACE TOPOLOGY WITH WORKSPACE MODELS

A Dissertation

by

READ MULLAN SANDSTRÖM

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Nancy M. Amato
Committee Members,	Suman Chakravorty
	Dylan Shell
	Dezhen Song
Head of Department,	Scott Schaefer

May 2020

Major Subject: Computer Science

ABSTRACT

Motion planning is an important problem in robotics which addresses the question of how to transition an actor between states in an environment subject to obstacles, kinematic, and other constraints. Exact motion planning is a computationally hard problem, which has prompted the popularity of sampling-based approaches. Algorithms in this family operate in the configuration space of a robot. This abstraction represents the space of possible poses, where feasible poses are termed valid. A solution to the planning constitutes a curve in the valid subset of configurations space, known as the free space.

However, computing an explicit representation of the free space is an intractable problem, which renders sampling-based planners blind to its structure. This requires the algorithms to discover the space's topology by randomized sampling and validity checking. We observe that when planning for physical robots, there are frequently strong ties between the physical environment or workspace and the free subspace corresponding to the translational degrees of freedom. When such a relation exists, we can leverage information about the topology of workspace to provide direction to the search in configuration space, thereby significantly reducing the search domain. We present two algorithms which realize this objective.

The first, Dynamic Region sampling, employs a topological skeleton of workspace to define the valid paths through the physical environment. This focuses the planner's generation of new samples within specific regions that travel along the skeleton, just ahead of the frontier of known valid configurations. This directs the search for a free space path along the known possibilities in workspace. The regions thus form both a guide for the planner and a mechanism of representing progress in covering the workspace.

The second, Topological Nearest-Neighbor Filtering, uses a cell decomposition of the workspace as a means of mapping configurations to neighborhoods in free space. This mapping supports rapid locations of other configurations in neighborhoods that are nearby through connected workspace. This builds a model of connectivity into the nearest-neighbor process, which is a critical compo-

nent in determining how to attempt connecting known configurations with local plans. The filter both expedites the nearest-neighbor operation and filters out obviously poor choices, promoting a higher rate of successful connection.

We show how these methods can improve the robustness and efficiency of sampling-based motion planners in problems where a robot must traverse a complex workspace. We demonstrate that the methods work synergistically with each other to guide both exploration of free space and connection of the roadmap representation, promoting fast feasibility planning in many difficult problems.

DEDICATION

To Elizabeth Anne, you are always in my thoughts.

ACKNOWLEDGMENTS

No work of science is truly an individual endeavor. We stand not only on the shoulders of those who came before us, but also on the supporting players who nurture and guide us through the other aspects of our lives. Here I would like to briefly acknowledge the contributions of my supporters, without whom this work would not have been possible.

To my advisor Nancy, thank you for creating a multitude of opportunities for me to grow as a scholar of computer science and teaching me how to make the most of each one.

To my committee members Suman Chakravorty, Dylan Shell, and Dezhen Song, thank you for your penetrating questions which helped shape this line of research.

To Jory Denny, thank you for suffering my countless argumentative questions and for setting me down the path towards excellence in software engineering.

To Shawna Thomas, Daniel Tomkins, Diane Uwacu, Irving Solis, James Motes, and Andrew Bregger, thank you for making the lab a great place to work and learn.

To my sister Kelly, thank you for jumping at every opportunity to help, both with my children and my graduate work.

To my children Adeney, Ivess, and Lincoln, thank you for growing into wonderful people and bringing peace to my heart.

To Lisa and David Forson, thank you for supporting our family and being great grandparents for our littles.

To my parents Robert and Jo Ellen, thank you for your endless support and for providing an ideal foundation of values and training for building my life.

To my wife Christin, thank you for standing by me through this journey, being generally awesome, and making me feel like I've won the game of life.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professors Nancy Amato, Dylan Shell, and Dezhen Song of the Department of Computer Science and Engineering and Professor Suman Chakravorty of the Department of Aerospace Engineering.

All implementations for this work were carried out by the student in the Parasol Motion Planning Library provided by Professor Nancy Amato.

The work on Dynamic Region RRT, published in the Proc. of the Workshop on the Algorithmic Foundations of Robotics, 2016, was a joint project with Jory Denny. Experimentation was assisted by Andrew Bregger.

Experimentation for the work on Topological Nearest-Neighbor Filtering, published in the IEEE International Conference on Robotics and Automation, 2018, was assisted by Andrew Bregger and Ben Smith.

Experimentation for the work on Dynamic Region sampling with PRM was assisted by Diane Uwacu and James Motes.

All other work conducted for the dissertation was completed by the student independently.

Funding Sources

This work was supported by a research assistantship and NFS awards CCF-1423111 and EFRI-1240483.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Motion planning is the problem of determining a valid trajectory which moves an object from a start configuration to a goal configuration. In the context of robotics, the object is a robot or manipulated object, and validity refers to both avoiding collisions and respecting the dynamic and kinematic constraints of the object's physical mechanisms. Motion planning is an important component of a robotic system that determines what actions should be taken to transition the robot from one physical state to another. This is necessary for planning all kinds of motions, from moving between rooms to grasping objects.

Exact motion planning has been shown to be PSPACE-hard [1] and is thus considered intractable. The best-known complete planner is Canny's roadmap algorithm [2], which is singly exponential in the degrees of freedom (DOFs) of the robot. To solve motion planning problems in reasonable time, research has focused on sampling-based algorithms. Sampling-based motion planners trade completeness for probabilistic completeness and a large gain in expected efficiency. Probabilistic completeness describes a weaker completeness property where a sampling-based algorithm is complete in the limit of infinite samples; it is useful in establishing that a given algorithm has a non-zero probability of solving any particular problem.

Sampling-based planners gain efficiency by avoiding an explicit construction of the configuration space, or $\mathcal{C}_{space}$, which is an abstract space of all possible configurations of the robot [3]. $\mathcal{C}_{space}$ is a useful abstraction for planning because particular poses of the robot may be represented as points, and motions may be represented as paths. At the same time, $\mathcal{C}_{space}$ is inconvenient because a direct representation of the obstacles is not feasible. The core methods of sampling-based planning thus only consider spatial information while collision testing a particular configuration of the robot against itself and the environment obstacles which define the workspace for a particular problem. This renders the planner effectively blind to the topology of the space: it has information about a few points and paths, but no notion of the shape of the valid subset known as the free space or $\mathcal{C}_{free} \subset \mathcal{C}_{space}$. Without such information, the search for a path in $\mathcal{C}_{free}$ must rely purely on ran-

domly sampling appropriate configurations to lead roadmap extension in a productive direction. However, the probability of this occurring drops dramatically with the problem difficulty. This has been described as the narrow passage problem [4].

Since collision-free validity is defined in terms of the workspace, the workspace is a very important component of virtually all robot motion planning problems. Prior work has shown that information regarding the workspace can be useful in solving many kinds of planning problems, especially where the workspace geometry is complex or contains narrow passages [5, 6, 7]. Such heuristics build on the observation that the workspace geometry is strongly correlated with the subset of $\mathcal{C}_{space}$ which describes the robot's translational DOFs.

Any solution to a motion planning problem must pass through some part of each subspace: a guidance mechanism which steers a planner through subspaces effectively provides a partial solution. We observe that the topology of workspace describes such partial solutions, and can be used to approximate information about the topology of $\mathcal{C}_{free}$. The goal of this work is to leverage the topology of the workspace (a particular subspace of $\mathcal{C}_{free}$ in which the obstacle boundary is known) to approximate the connectivity of $\mathcal{C}_{free}$ in sampling-based planning. We present two algorithms, Dynamic Region sampling and Topological Nearest-Neighbor Filtering, which employ models of workspace to achieve such an approximation.

Dynamic Region sampling employs a topological skeleton of workspace to define the valid paths through the physical environment. It focuses the planner's generation of new samples within specific regions that travel along the skeleton, just ahead of the frontier of known valid configurations. This provides a coarse model of the translational subspace for the planner to follow, thereby limiting the search domain to the parts of $\mathcal{C}_{space}$ that might contain solution paths in $\mathcal{C}_{free}$.

Topological Nearest-Neighbor Filtering uses a cell decomposition of the workspace as a means of mapping configurations to neighborhoods in free space. For a given configuration q, this mapping supports rapid locations of other configurations in neighborhoods that are nearby through connected workspace. As workspace is a subset of $\mathcal{C}_{free}$, the located configurations are a superset of those which are near to q through connected $\mathcal{C}_{free}$. This builds a model of connectivity into the

nearest-neighbor process, which is a critical component in determining how to attempt connecting known configurations with local plans. The filter both expedites the nearest-neighbor operation and filters out obviously poor choices, promoting a higher rate of successful connection.

We show that these algorithms significantly improve sampling-based planners in problems where the workspace is relatively tight and complex. By following an approximate model of the $\mathcal{C}_{free}$ topology, the planner is able to focus effort on the relatively small portions of $\mathcal{C}_{space}$ that may produce a valid path. The net effect is to shrink the search domain to a much smaller relevant set for faster and more robust planning.

Portions of this research were previously published. The work on Dynamic Region sampling for RRT methods was published in part in Jory Denny's PhD thesis [8] and the proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR) 2016 [9]. The material on Topological Nearest-Neighbor Filtering with k nearest-neighbors appears in the proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2018 [10].

2. PRELIMINARIES AND RELATED WORK

For a movable object R such as a robot, a *configuration* of R fully specifies its position, orientation, and dependent component layout with respect to its environment E. The set of all configurations of R within E is the *configuration space* or $\mathcal{C}_{space}$ of R in the context of E [3]. The dimension of $\mathcal{C}_{space}$ is equal to the degrees of freedom (DOFs) of R. The set of *valid* (i.e. collision-free) configurations of R is referred to as the *free space* $\mathcal{C}_{free} \subset \mathcal{C}_{space}$, while the set of *invalid* configurations is referred to as the *obstacle space* $\mathcal{C}_{obst} \subset \mathcal{C}_{space}$.

Given a movable object R, an environment E, and start and goal configurations $q_s, q_g \in \mathcal{C}_{free}$, the *motion planning* problem is to find a valid trajectory τ:

$$\tau(t) : \mathbb{R} \to \mathcal{C}_{free} \mid t \in [0,1],\ \tau(0) = q_s,\ \tau(1) = q_g$$

Real robots always have some space of allowed controls U and a *dynamics model* such as $C(q,u) = \dot{q}$, which describes the change in configuration $\dot{q}$ which results from applying a control $u \in U$ to the robot at some initial configuration $q \in \mathcal{C}_{free}$. This is an example of a first-order dynamics model in which a robot can directly control its velocity; a more realistic model describes the output of C as an acceleration $\ddot{q}$ or higher-order derivative of q.

When dynamics are considered in the planning problem, we are often concerned with various derivatives of a configuration q in addition to q itself. The combination of a configuration q and one or more of its derivatives in this context is referred to as a *state* as in classical mechanics and control theory. The set of all states is correspondingly termed the *state space*.

A *holonomic* robot is one with no constraints on its change in state: it can instantly achieve any velocity $\dot{q}$ from any configuration $q \in \mathcal{C}_{free}$. This is an idealized model which essentially ignores the robot's dynamics to simplify the planning problem. Such robots have a state space which is identical to their configuration space. In contrast, robots with velocity or higher-order constraints (grouped together under the term *differential constraints*) are referred to as *nonholonomic*. Since

their dynamics models depend on derivatives of their state, such derivatives must be incorporated into their states. For a nonholonomic robot with constraints on n derivatives of its configuration, the state space must include all n derivatives and will have dimension equal to $n * \text{DOFs}$.

For nonholonomic robots, the motion planning problem is formulated with respect to the set of allowed controls since these are the inputs that the robot can accept. Let the full state space of the robot be denoted by X, which defines all valid (constraint-satisfying) states ignoring obstacles. For a given state $x \in X$, let $q(x) : X \to \mathcal{C}_{space}$ denote the robot's configuration at x. In this context, the start will be a state x_s, and the goal is usually relaxed to a region in state space X_g because it is often extremely difficult to arrive at an exact state. The problem is then to find a continuous sequence of controls u which produce a valid trajectory τ in state space:

$$u(t) : \mathbb{R} \to U,\ \tau(t) : \mathbb{R} \to X \mid t \in [0, 1],\ q(\tau(t)) : \mathbb{R} \to \mathcal{C}_{free},\ \tau(0) = x_s,\ \tau(1) \in X_g$$

2.1 Sampling-Based Planning

Sampling-based motion planning is an algorithmic paradigm which aims to efficiently construct a coarse map of configuration space to solve a motion planning problem. The core idea is to randomly generate samples in configuration space, check their validity, and attempt to connect valid configurations with simple *local plans*, which are short paths in free space. There are two major families of these algorithms, which are *graph-based* and *tree-based*.

The graph-based family of algorithms is descendent from the Probabilistic Roadmap, or PRM [11] method. These methods construct a graph roadmap in free space by connecting randomly sampled configurations to their k nearest neighbors. The resulting roadmap can be queried for a path from start to goal using a graph search algorithm such as Dijkstra's [12] or A* [13]. PRM methods are intended for settings where the roadmap will be used for multiple queries in the same environment. They provide good connectivity and often encode more than one path between particular configurations, which makes them particularly well suited to multiple query problems with multi-

ple robots. However, PRMs can be difficult to apply to nonholonomic robots because constructing a local plan between two states requires a *steering function*, which is not readily available for many robots.

The tree-based family of algorithms is descendent from the Rapidly-Exploring Random Tree, or RRT [14] method. RRT methods work by sampling a random configuration q_{rand} and extending the nearest existing roadmap configuration q_{near} towards it for a finite distance. This incrementally builds a tree roadmap in free space which can be queried for a path in the same manner as their PRM counterparts. A subsequent extension showed that building a tree from both the start and goal configurations (commonly referred to as bi-directional RRT) can yield significant performance benefits in many scenarios [15]. Since it does not rely on the ability to exactly connect q_{near} to q_{rand}, RRT can easily handle differential constraints by searching the control space for the best control to steer towards q_{rand} from q_{near} [16].

2.1.1 Probabilistic Completeness

All sampling-based planners suffer from a common problem, which is the lack of completeness. A sampling-based planner would need to attempt sampling every possible point in configuration space to determine that no solution exists, and may need to do this in the worst case if there is only one valid solution and it is the last one attempted. This leads to the notion of *probabilistic completeness*: in the limit where the sampled configurations cover the entire configuration space, a sampling-based planner will discover a valid path if one exists. In practice, few problems are so difficult that only one valid path exists, and sampling-based planners can frequently identify even very obscure paths with appropriate guidance mechanisms.

2.1.2 The Narrow Passage Problem

A common problem with uniform sampling is a relatively low probability of identify narrow regions of free space, which are known as *narrow passages*. When such areas are necessary for a valid trajectory, sampling-based planners require some form of biasing mechanism efficiently discover them.

Two approaches for this which work purely in configuration space are the obstacle-based PRM (OBPRM) [17] and TogglePRM [18]. OBPRM works by sampling line segments in configuration space to search for pairs of configurations which lie on either side of the obstacle boundary. The free configuration in this pair is necessarily very close to the obstacle. This strategy works very well for identifying configurations in tight spaces like narrow passages. TogglePRM similarly leverages the obstacle space by simultaneously mapping it alongside the free space. Failed local plans in either space identify some area of the alternate space: narrow passages are easily discovered by this method when local planning in the obstacle space.

2.1.3 Workspace-Guided Methods

In many robotics problems, the free workspace is highly correlated with the translational components of the free configuration space. Several methods have leveraged this observation to use the free workspace as a guide for generating samples. The Workspace Importance Sampling (WIS) [5] method biases configuration sampling based on a cell decomposition of the workspace. Cells are analyzed to determine the probability that they are located within a narrow passage; this probability is then used as a bias for preferentially sampling within a given cell. A similar method employs a watershed test to identifies constriction in the workspace and increases the sampling density within [6]. Another class of methods are the medial-axis based planners, which employ a medial axis of the workspace to generate samples that are far from obstacles [19, 20, 21].

A notable workspace-guided technique which does not merely bias PRM sampling is the RRT-based method Synergistic Combination of Layers of Planning (SyCLoP) [7]. SyCLoP employs a discrete search over a workspace decomposition to identify a potential workspace solution path; it then selects a roadmap configuration near the relative frontier within this path for expansion. This method was designed for random control propagation with nonholonomic systems, however it can easily be extended to holonomic robots and/or best-control selection as in our work [9] (described in Chapter 3).

2.1.4 Asymptotically Optimal Planning

Another area of interest in sampling-based motion planning is the study of *asymptotically-optimal* (abbreviated by AO) planning algorithms. These techniques continue to refine the solution after an initial trajectory is identified, and asymptotically converge to an optimal solution. Variations of the major algorithm families have been extended to support AO, and are denoted by attaching a star to the algorithm name (PRM*, RRT*) [22]. Since no robotics application can afford unlimited sampling time, many authors refer to these methods as 'anytime' planners to indicate that the current best solution may be extracted any time after the initial discovery of a feasible path.

However, these variants require steering functions to support nonholonomic robots, and are thus not applicable to systems without a known steering function. To combat this difficulty, the Stable Sparse Tree or SST method provides an RRT-like algorithm which achieves AO behavior without a steering function [23]. This is achieved by enforcing sparseness of the produced tree and extending only from the lowest-cost configuration within a sparsified region of state space. Although tuning the sparseness is non-trivial, the SST is theoretically important in being the only algorithm to provide AO without a steering function.

2.2 Nearest-Neighbor Algorithms

In all sampling-based planning algorithms, the two most expensive components are collision-checking and nearest-neighbor location. While several techniques for assuaging the cost of collision-checking have been investigated [24, 25], in the present work we focus on the nearest-neighbor problem.

The nearest-neighbor problem is to locate the k nearest neighbors from a set of known points Q to some query point q, usually in a high-dimensional space. Motion planning algorithms solve this problem repetitively when determining which configurations to connect in various contexts. Several computational techniques have been developed for locating exact and approximate nearest-neighbors.

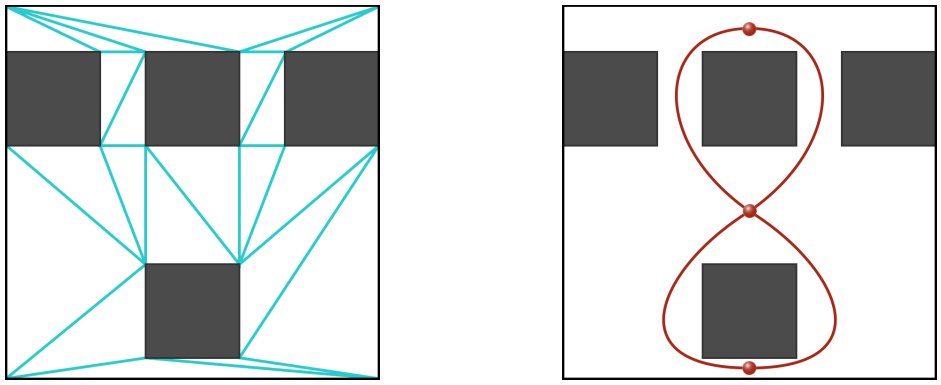
The simplest approach to the nearest-neighbor problem is to compare each candidate point in Q with the query point q; the closest k points in Q are then returned as the nearest neighbors. This approach is known as brute force or linear scan. While simple, it takes $\mathcal{O}(|Q| \lg k)$ time. In motion planning algorithms, Q represents the set of roadmap configurations. Each $q \in Q$ must be connected to at least one other configuration to be useful, so a nearest neighbor problem must be solved for each such configuration. Using linear scan for this results in a complexity of $\mathcal{O}(|Q|^2 \lg k)$ over the entire planning problem.

A more sophisticated approach is to use a k-d tree, which can locate points in logarithmic time [26]. A popular k-d tree method for motion planning also allows relaxing the exactness property for increased efficiency [27]. A later work re-iterates on this method to develop a partitioning strategy which better respects the nuances of the orientation components [28]. Other structures such as metric and cover trees have also been suggested, although empirical evidence suggests that these are not significantly faster than k-d trees in problems of moderate size and dimension [29].

However, other researchers have noted that k-d trees perform little better than brute-force when the problem dimension is moderate to large [30]. Such difficulties have spurred investigation in approximate nearest neighbor methods, which aim to trade a relatively small sacrifice in accuracy for a larger gain in computational speed.

Locality-sensitive hashing (LSH) is a class of approximate methods which attempts to bucket similar samples together with some form of hashing scheme, and has been applied in both machine learning [31, 32] and motion planning [30]. An alternative approximate method from the motion planning realm is distance-based projection onto Euclidean space (DPES) which uses a projection from configuration space to a lower-dimensional Euclidean space before computing the neighbors [33].

Another notable exact approach which has been applied in motion planning contexts is the geometric nearest-neighbor access tree (GNAT) [34]. This method identifies all nearest neighbors within a given radius of the query point (as opposed to the k nearest neighbors within any distance). It employs a hierarchical Voronoi decomposition of a metric space to filter out large segments of



(a) A cell decomposition (b) A topological skeleton

Figure 2.1: The two types of workspace models used in this work. Obstacles are shown in gray, free workspace in white, and model components in color.

the domain which exceed the search radius. It was designed for cases where measuring distance is expensive and has been applied in motion planning with a deep-learning based swept-volume distance metric [35].

2.3 Workspace Models

Throughout this work we will rely on models of the workspace which describe its topology. Two types of models will be used, which are cell decompositions and topological skeletons.

2.3.1 Cell Decompositions

A cell decomposition is a partitioning of volume into a set of discrete cells (Fig. 2.1(a)). Cell decompositions are a well-studied area of computational geometry and are covered in standard texts [36]. In addition to partitioning the volume, a cell decomposition should also describe the adjacency relationships between the cells, usually in the form of an undirected graph. This structure effectively forms an atlas of the volume with cells as the component charts.

Decompositions usually produce convex cells. Common choices are grids, and triangulations (also called tetrahedralizations for three-dimensional volumes). Grids necessitate some overlap with the obstacle space while triangulations do not; the latter can precisely represent a free space which is presented as a piecewise linear complex. Tetrahedralizations also share this property,

although it may be necessary to introduce additional vertices called Steiner points to produce a valid result [37].

In this work we will refer to cell decompositions of the free workspace (i.e. physical space without obstacles) as *workspace decompositions* or simply decompositions.

2.3.2 Topological Skeletons

A topological skeleton or topological graph is a 1-complex representation of a space in which vertices map to points in the space and edges map to paths between their source and target [38] (Fig. 2.1(b)). There are many ways to produce skeletons with different properties, such as a medial axis [36], Reeb graph [39], or mean-curvature skeleton [40]. Concatenations of the edge paths describe the set of homotopy classes for paths through the volume.

In motion planning, skeletons are useful as compact representations of a space where vertices represent open volumes and edges represent junctions between them. We will refer to skeletons of the workspace as *workspace skeletons* or simply skeletons.

2.3.3 Inner Distance

For parts of our work, we will require a measurement of the shortest-path distance between two cells in a workspace decomposition without entering the obstacle space. This is known as the inner distance [41] or euclidean shortest path distance. We will prefer the former term to avoid confusion with the general euclidean distance. An exact computation of this distance is known to be NP-Hard [42], but bounded approximations exist which can estimate the distance in polynomial time [43].

3. DYNAMIC REGION SAMPLING*

All sampling-based planners must sample configurations to explore $\mathcal{C}_{space}$, but it is not necessary that they do so in a uniform random fashion. Certainly some areas of $\mathcal{C}_{space}$ are more likely worth the computational resources required to explore them than others; the volumes inside obstacles are clearly outside of $\mathcal{C}_{free}$, for instance. We observe that a workspace skeleton describes valid paths through the free workspace, which is closely tied to the translational subspace of $\mathcal{C}_{free}$. This suggests that sampling near the vertices and edges of a workspace skeleton is more likely to produce configurations on a valid path through $\mathcal{C}_{free}$ than uniform random sampling.

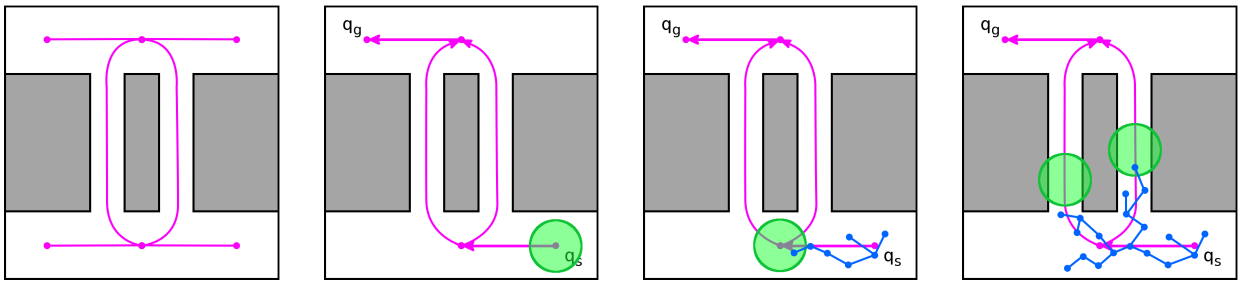
Dynamic Region sampling is a paradigm for biasing the generation of new configurations along regions that traverse a workspace skeleton, which focuses sample generation on more promising locations. Additionally, it provides a mechanism for tracking the algorithm's progress in extending the roadmap across the skeleton and further biases the sampling distribution to favor regions just ahead of the current roadmap frontier. We show how this paradigm can be applied to RRT planners for fast feasibility planning, and to PRM planners for building roadmaps with good connectivity.

3.1 Dynamic Region RRT

Dynamic Region sampling can be employed to guide an RRT planner along a topological skeleton of the workspace [9]. The method assumes a topological skeleton S of the workspace W as input (Fig. 3.1(a)), which encodes the valid paths between all portions of W. However, we observe that the goal of RRT is feasibility planning, and it is not necessary to consider the entire skeleton.

We first prune and direct the skeleton to represent the query at hand by removing superfluous components. We identify the skeleton vertices $v_s, v_g \in S$ which are nearest to the planning problem's start and goal respectively, and prune the skeleton of components which lead away from a

*Portions of this chapter describing the core concepts and application to RRT planners are reprinted with permission from [9] J. Denny, R. Sandström, A. Bregger, and N. M. Amato. "Dynamic Region-biased Rapidly-exploring Random Trees," in *Alg. Found. Robot. XII*, Springer, 2020. (WAFR 2016).



(a) Workspace and Skeleton (b) Initialization (c) Tree Growth (d) Multiple Regions

Figure 3.1: A sketch of dynamic region RRT. (a) A workspace skeleton (pink) is generated. (b) The skeleton is directed and pruned for a given query, and the first region (green) is initialized. (c) The region advances as the tree (blue) extends. (d) Multiple regions are generated as the tree passes a skeleton junction point with two outbound edges. Reprinted with permission from [9], © 2020 Springer.

possible (workspace) path from v_s to v_g (Fig. 3.1(b)). This is accomplished in two phases: first, a single-source shortest paths algorithm is executed from v_s to identify the set of edges which lead away from the start. Second, we execute a backward breadth-first search from v_g (tracing the inbound edges) to identify all ancestors of v_g in the search tree from v_s; any non-ancestors are removed from S. The remaining edges in S are then known to lie along a path from v_s to v_g, which represent the relevant portions of workspace for this particular query. This pruned skeleton is computed once prior to RRT execution for use as a sampling guide. The final initialization stage is to create a set of sampling regions on each edge outbound from v_s to begin the guidance process.

On each iteration of RRT a random configuration q_{rand} is generated from some sampling boundary, which is usually all of $\mathcal{C}_{space}$. In Dynamic Region RRT, we employ a weighted selection to choose either one of the sampling regions or the entire $\mathcal{C}_{space}$as the sampling domain. This is initially with equal probability, but quickly leans towards a strong favor of sampling regions which are productive in extending the roadmap. When a sampling region r is selected, we sample our growth target q_{rand} from r and attempt to extend towards it from the nearest configuration in the roadmap, q_{near}. This causes the tree to grow towards the area of workspace covered by r.

If the tree is extended into a sampling region r, we advance r along its skeleton edge until either

r is clear of the newly extended configuration or the end of the end is reached (Fig. 3.1(c)). In the later case, r has arrived at a new skeleton vertex $v \in S$, which indicates that the roadmap tree now covers r's skeleton edge. We create additional sampling regions for each outbound edge from v which have not already been generated to continue guiding the roadmap through the next segments of workspace (Fig. 3.1(d)). This advancement strategy keeps the sampling regions just ahead of the roadmap frontier, effectively tracking the planner's progress in covering the region around the skeleton. Biasing samples just ahead of the roadmap forces the planner to continue expanding the roadmap outward rather than 'infilling', which occurs when the growth target is generated within a region already well-covered by the roadmap.

As the sampling regions are traversing edges of a workspace skeleton rather than a $\mathcal{C}_{space}$ skeleton, it is possible that there is no valid path in $\mathcal{C}_{space}$ which corresponds to a given workspace edge. To address this, we adaptively tune the probability of selecting a given region r based on the recent success rate of generating samples within r. Regions that are performing well will be preferentially selected to exploit the ease of planning through the corresponding workspace volume. Regions which are not performing well will have a lower probability of selection. This feature automatically tunes the selection probabilities to favor exploiting good regions and exploring when none are known. It additionally allows the method to gracefully degrade to a standard RRT in the event that the workspace does not provide a reasonable guide for solving the given problem.

We retain the probability of selecting the full $\mathcal{C}_{space}$ as the sampling region to ensure probabilistic completeness. Unlike the sampling regions, the probability of selecting the entire $\mathcal{C}_{space}$ is not updated with respect to the rate of successful extension. This is because the update represents a metric for how well we have exploited a region recently, and we do not consider the $\mathcal{C}_{space}$ as an exploitable region. Rather, sampling from it represents a choice of exploration which broadens the search space when the dynamic sampling regions are performing poorly.

In applications for nonholonomic robots, we additionally need to be aware of the robot's momentum and control capabilities. For a realistic model of such a robot (i.e. a quad copter), it is often quite difficult to extend towards a growth target q_{rand} which is too close to the nearest-

Algorithm 1 Dynamic Region RRT

function DYNAMICREGIONRRT(Environment e, Configuration q_{start}, Configuration q_{goal})
    Initialize
    $S \leftarrow$ COMPUTEWORKSPACESKELETON(e)
    $S \leftarrow$ PRUNE(S, q_{start}, q_{goal})
    $R \leftarrow$ INITIALIZEREGIONS(S, q_{start})
    $G = \{G_V, G_E\} \leftarrow \{\varnothing, \varnothing\}$)
    Main loop
    while $\neg$done
        $r \leftarrow$ SELECTREGION(R, e)
        $q_{rand} \leftarrow$ SAMPLE(r)
        $q_{near} \leftarrow$ NEARESTNEIGHBOR(q_{rand}, G_V)
        $q_{new} \leftarrow$ EXTENDTREE(q_{near}, q_{rand})
        if $q_{new} \neq \varnothing$
            $G_V \leftarrow G_V \cup q_{new}$
            $G_E \leftarrow G_E \cup (q_{near}, q_{new})$
            ADVANCEREGIONS(q_{new}, S, R)
    return G
function ADVANCEREGIONS(Configuration q_{new}, Skeleton S, Regions R)
    Check each region for contact with q_{new}
    for all Region $r \in R$
        while $q_{new} \in r$
            Check for end of edge
            $p \leftarrow r$.GETNEXTSKELETONEDGEPOINT( )
            if $\nexists p$ ▷ r has reached the end of the edge
                $R \leftarrow R \setminus r$
                CREATEREGIONS(r.edge.target)
                continue
            r.center $\leftarrow p$

neighbor q_{near}. This problem occurs when the robot's momentum cannot be quickly overcome by any control, so such an extension is likely to produce an erratic path in an attempt to steer from q_{near} to q_{rand}. A simple fix for this is to relax the requirement for advancing a region: rather than requiring the newly generated configuration q_{new} to be fully contained within the sample region r, we instead advance r if q_{new} lies partially within r. This has the effect of prematurely advancing the sampling regions so that the sampled growth targets lead the tree by a greater distance, which improves the likelihood that the extension will be fruitful. The level of relaxation is parameterized with respect to the robot and region radius; robots which can achieve large momentums benefit from more relaxation, while robots with sufficiently strong controls do not require as much.

Dynamic Region sampling adds little overhead to standard RRT growth. Additional operations include selecting, advancing, creating, and deleting regions. The maximum number of active regions $|R|$ is $O(|S_E|)$, at most linear in the number of edges in the skeleton S. Checking for region advancement takes $O(|R|)$ operations to check each active region. Over the whole problem, advancing the regions along the edges S_E is bounded by the number of sub points $P_e \in S_E$, for a total cost of $O(|P_e|)$. However, these are extremely pessimistic upper bounds. In most scenarios, the number of active regions is much smaller than the number of skeleton edges, and the number of uncovered edges drops monotonically during execution.

3.1.1 Validation

We demonstrate Dynamic Region RRT on holonomic robots in five environments, including `MazeTunnel`, `LTunnel`, `Garage`, `GridMaze4`, and `GridMaze8` as shown in Figure 3.2. We also evaluate performance on the `LTunnel` with nonholonomic robots, comparing against RRT and SyCLoP, to show that Dynamic Region sampling can benefit nonholonomic systems. These environments contain narrow passages in the workspace that are correlated with valid paths in $\mathcal{C}_{space}$, and are thus good exhibitions of the planner's ability to exploit that correlation. The difficulties in each environment vary. `MazeTunnel` contains false passages, `LTunnel` incorporates narrow entrances, `Garage` has multiple homotopy classes, and `GridMaze` environments have long, winding paths in a cramped tunnel that constrain the robot's rotational DOFs.

The robots in all cases are 6 DOF rigid bodies. The nonholonomic experiment uses a fully actuated robot with a 1:2 ratio of random vs. best control selection and a 1:2 ratio of fixed vs. variable timestep for extension [44].

We compared Dynamic Region RRT (DR-RRT) against RRT [44], Dynamic-Domain RRT [45], Synergistic Combination of Layers of Planning (SyCLoP) [7], and the Probabilistic Roadmap (PRM) approach using Workspace Importance Sampling (WIS) [5]. We selected these representative methods for their similarities to our approach. Dynamic-Domain RRT, SyCLoP, and WIS use adaptive sampling distributions. SyCLoP and WIS employ pre-computations to decompose the workspace. SyCLoP additionally uses a search over the decomposition graph to guide an RRT planner.

3.1.1.1 Experiment Setup

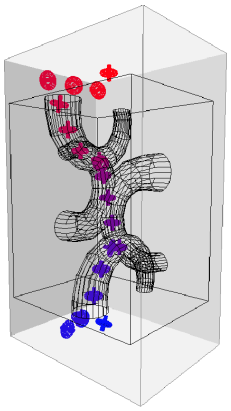
All methods were implemented in a C++ motion planning library developed in the Parasol Lab at Texas A&M University. All experiments were executed on a laptop running Fedora 23 with an Intel® Core™ i7-6500U CPU at 2.5 GHz, 8 GB of RAM, and the GNU gcc compiler version 5.3.1.

Each experiment ran until the query was solved (success) or 20,000 nodes had been added to the roadmap or 20 minutes had elapsed (failure). We performed 35 trials for each experiment and removed the fastest and slowest run from each. Success rates are shown in Table 3.1, and average run times and standard deviations are shown in Figures 3.3 and 3.4. Times are reported over all runs.

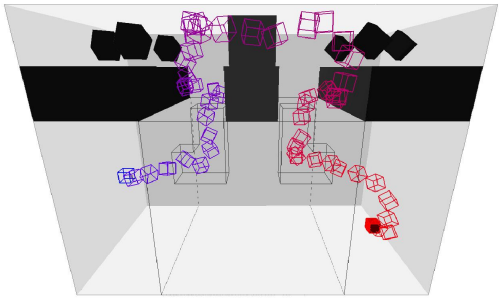
Skeletonization was considered as a pre-processing step and not included in the run time. These experiments use a Reeb graph skeleton, for which construction took an average 3.5 seconds to build in `MazeTunnel`, 0.7 seconds in `LTunnel`, and 2.3 seconds in `Garage`.

3.1.1.2 Analysis

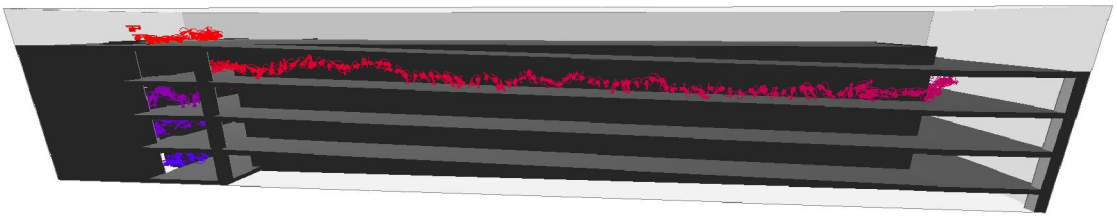
As Figure 3.3 shows, Dynamic Region RRT exhibits faster planning time as compared with the other methods for the holonomic problems. The improvements over the closest competitors are significant, with $p_{\text{val}} = .0083$ against RRT and $p_{\text{val}} = .0175$ against Dynamic-Domain RRT in



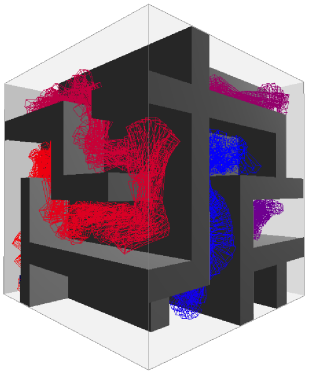
(a) `MazeTunnel` (toroidal plus)



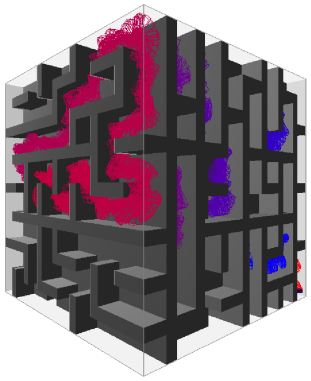
(b) `LTunnel` (box)



(c) `Garage` (helicopter)



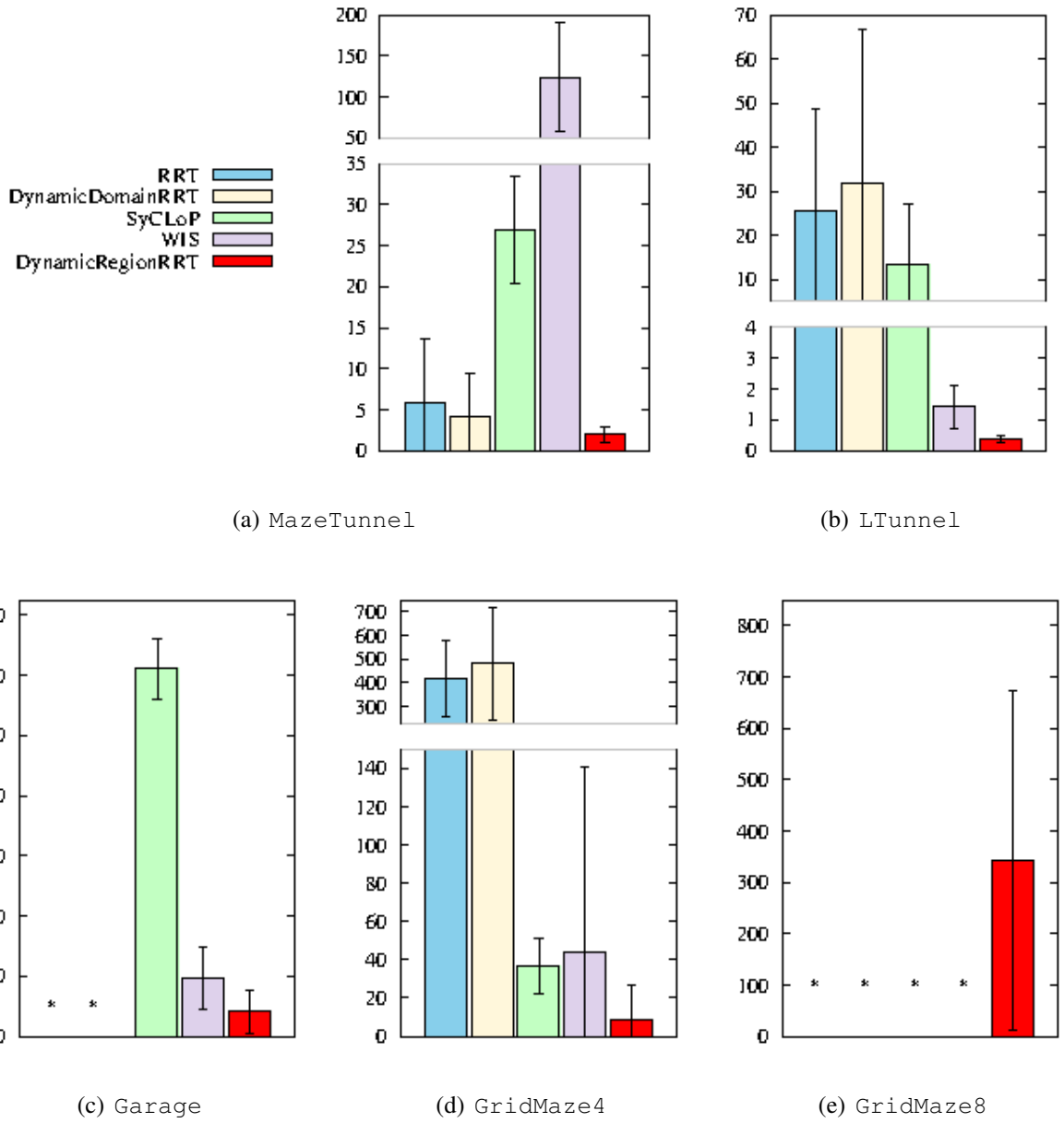
(d) `GridMaze4` (stick)



(e) `GridMaze8` (stick)

Figure 3.2: Test environments for Dynamic Region RRT. The `Garage` contains multiple alternative paths. The `GridMazes` are 3D mazes with internal pathways and tunnels throughout. The robots in all cases are 6 DOF rigid bodies. In the `GridMazes`, the robot length is equal to the tunnel width to constrain rotational motion. Reprinted with permission from [9], © 2020 Springer.

`MazeTunnel` (using the student's unpaired t-test). The closest competitor in the other environments were slower with $p_{\text{val}} < .0001$ (WIS in `LTunnel` and `Garage`, SyCLoP in `GridMaze4`). Neither RRT nor Dynamic-Domain RRT was able to solve the `Garage` problem. WIS showed



(a) `MazeTunnel` (b) `LTunnel`

(c) `Garage` (d) `GridMaze4` (e) `GridMaze8`

Figure 3.3: Evaluation of Dynamic Region RRT in holonomic problems. Plots show the average planning time over 33 trials. Error bars indicate standard deviation. A * indicates that none of the trials solved the query. Reprinted with permission from [9], © 2020 Springer.

strong performance in `LTunnel` and `Garage`, but lacked efficiency in `MazeTunnel` and consistency in `GridMaze4`. Only DR-RRT was able to solve the large `GridMaze8`, which pushes the algorithm to the limits of what it can handle with high reliability.

Planner	DR-RRT	RRT	Dynamic-Domain RRT	SyCLoP	WIS
`MazeTunnel`	100%	100%	100%	100%	100%
`LTunnel`	100%	100%	100%	100%	100%
`Garage`	100%	0%	0%	100%	100%
`GridMaze4`	100%	100%	100%	100%	100%
`GridMaze8`	76%	0%	0%	0%	0%
`LTunnel` (nonholonomic)	90%	24%	-	97%	-

Table 3.1: Evaluation of Dynamic Region RRT in holonomic problems. The table shows the success rates in each experiment. Reprinted with permission from [9], © 2020 Springer.

Compared with the other RRT methods, DR-RRT's topological guidance leads to more productive samples and thus smaller roadmaps. This results in faster neighborhood finding tests and faster overall execution. Despite SyCLoP's mechanism for expediting neighborhood finding, our guidance mechanism showed significantly lower per-extension cost, which more than compensated for this in the holonomic trials. WIS generally produced the smallest roadmaps, but often took a long time to generate its samples and spent much more time connecting them due to being a PRM method.

In the nonholonomic experiment (Figure 3.4), DR-RRT was faster than RRT ($p_{\text{val}} < .0001$) and slower than SyCLoP ($p_{\text{val}} = .0002$) in `LTunnel` with high confidence. While DR-RRT improves over RRT, it doesn't perform as well as SyCLoP for this problem. This is because SyCLoP's neighborhood finding expedition offers more benefit in problems with larger roadmaps (like nonholonomic problems), and because it doesn't always expand from the front of the tree, which helps prevent it from getting stuck when the leading nodes in the tree are difficult to extend from.

3.2 Theoretical Considerations

The skeleton must be a deformation retract [46] of the free workspace to ensure that it adequately represents the environment. This implies that every point in the free workspace can be mapped onto the skeleton, i.e., it represents the entire workspace and preserves its topology. This is important because every workspace homotopy is a generalization of one or more $\mathcal{C}_{space}$ homo-

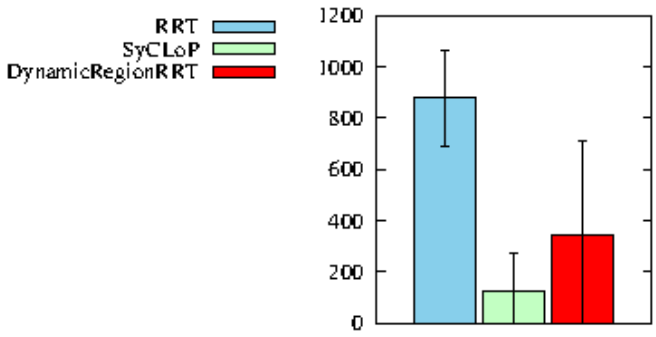


Figure 3.4: Evaluation of Dynamic Region RRT in nonholonomic problems. Plot shows the average planning time over 33 trials. Error bars indicate standard deviation. Reprinted with permission from [9], © 2020 Springer.

topy classes (i.e., with a projection of a $\mathcal{C}_{space}$ homotopy into workspace using only the positional DOFs). We thus require the skeleton to be complete to avoid missing any $\mathcal{C}_{space}$ homotopy classes.

Some problems exhibit only a partial correlation between $\mathcal{C}_{space}$ and workspace. One example is manipulation planning, where the end-effector's path is strongly correlated with the environment, but the movement of the rest of the robot may not be. In such cases, a skeleton could be used to guide sampling for the correlated component. This would additionally require a sampling method that is able to place the correlated component first and subsequently sample the remaining DOFs, such as reachable volumes [47] or cyclic coordinate descent [48].

Unfortunately, not all problems exhibit a useful correlation between the workspace and $\mathcal{C}_{space}$ topology. In problems such as the alpha puzzle, the use of a workspace skeleton may not be sensible. However the underlying idea may still apply, which is to use regions and topological analysis in low-dimensional manifolds of $\mathcal{C}_{space}$ to explore $\mathcal{C}_{free}$. We have so far considered the translational subspace as this manifold, but one could feasibly design alternatives for other classes of problems.

Dynamic Region sampling can be expected to work well when the union of all sampling regions covering the skeleton points contains a path-connected volume in $\mathcal{C}_{free}$. Formally, define the metric space $M_C = (\mathcal{C}_{space}, D)$ where D is the euclidean metric and $M_W = (W, T)$ where W is the workspace and T is translational distance. Let $\beta_{cr}(q)$ be a ball in M_C of radius r centered at

$q \in \mathcal{C}_{space}$, and let $\beta_{wr}(p)$ be a ball in W of radius r centered at $p \in W$. Let $\tau(q) : \mathcal{C}_{space} \rightarrow W$ represent the mapping between the translational subspace of $\mathcal{C}_{space}$ and the robot's reference point in W. Then for any ball $\beta_{wr}(p) \in W$, the inverse map $Q(p) = \tau^{-1}(\beta_{wr}(p)) \in \mathcal{C}_{space}$ describes a hypercylinder in $\mathcal{C}_{space}$ which is a ball in the translational subspace centered on $\tau^{-1}(p)$. Therefore $\beta_{cr}(\tau^{-1}(p)) \subset Q(p)$, meaning that a ball of radius r in M_W encompasses a superset of the corresponding ball of radius r in M_C. The union of all possible sampling regions along the skeleton U thus represent a union of hypercylinders in $\mathcal{C}_{space}X$; in the case where U is path-connected, X will be also.

To ensure that U contains a path-connected volume in $\mathcal{C}_{free}$, it remains to show that the intersection $Y = X \cap \mathcal{C}_{free}$ is path-connected. A general argument for this is not possible due to the wide variety of choices for the skeleton, robot, and environment. For example the workspace and skeleton may be disjoint (in which case no planner can succeed in completely connecting the space), the skeleton may be badly positioned (resulting in disjoint components for Y), or the robot may be too large to traverse into certain regions of workspace (again resulting in disjoint components for Y). As such, this description serves as a characterization of when the method can produce a good coverage of W and $\mathcal{C}_{free}$ rather than a statement that it will always do so.

There are at least two cases where one can be assured that Y is path-connected. The first is when the robot is a free-body with maximum radius less than or equal to some value ρ and the skeleton has clearance greater than or equal to ρ everywhere. This holds for some common cases where floor-dwelling robots must perform tasks in large but reasonably uncluttered spaces. A second case is where there is a valid configuration at each skeleton point and a valid local plan between them: this represents the case where one knows that the robot has a valid maneuver for all localities, and shows a similar flavor to human intuition in collaborative planning [49]. A third case is for a point robot, which will always meet this criterion because its $\mathcal{C}_{space}$ is exactly the workspace. Non-rotational robots will similarly satisfy when the skeleton is path-connected in an 'inflated' workspace produced by the Minkowski sum of the robot with each obstacle.

In cases where Y is path-connected, a planner driven by Dynamic Region sampling is prob-

abilistically complete if the skeleton meets the definition of a deformation retract. This is true because a retract skeleton is visible to the entire $\mathcal{C}_{free}$, and a path between any two points can be formed by connecting each point to its nearest visible point on the skeleton. Adding sampling from the full environment ensures probabilistic completeness even when Y is not path-connected or the skeleton is ill-formed; however as these assumptions break down, we expect the performance of the region guidance to degrade correspondingly.

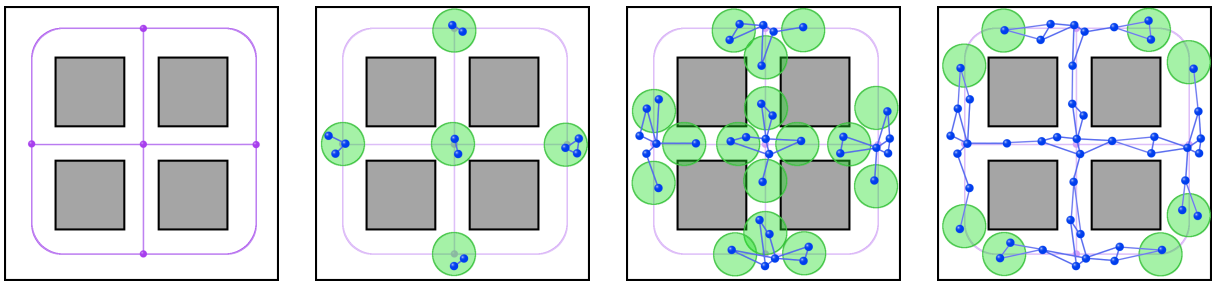
3.3 Sampling Regions and Clearance Awareness

We assume topological skeleton of the workspace $S = S_V, S_E$ as an input element, which may be any type with reasonable clearance properties. Throughout the algorithm, we define sampling regions at vertices and points on the edge paths. These regions may be given some fixed size, but we observe that an alternate strategy can be employed to leverage the known clearance in workspace.

At any point p in workspace, we can define a spherical sampling region r with clearance awareness by considering the available space to place the robot. Let the center be p and radius be defined as the clearance at p minus the robot's minimum radius from the reference point p_r defining its translational DOFs. Define r such that any samples generated will place p_r within the sphere. Such a region r inscribes the maximum region of workspace around p where a sample could be placed for p on the medial-axis.

In practice, few skeletons truly lie on the medial axis, and this sizing mechanism may preclude the generation of configurations at points where the skeleton's clearance is poor. To avoid this problem, a minimum radius should be selected for non-medial skeletons to account for the fact that the clearance is not uniform around the skeleton components.

When working with a skeleton on the medial-axis, we can employ pure clearance-based sizing to filter out regions of workspace that cannot accept the robot. This enables the planner to avoid obstructions with small holes such as chain-link fences without wasting effort on an unfruitful exploration. In this case, the skeleton can be pruned of low-clearance points as a pre-processing step. We note that for non-medial skeletons, the biasing of region selection based on success



(a) (b) (c) (d)

Figure 3.5: An illustration of Dynamic Region sampling with PRM. Obstacles are shown in gray. (a) The workspace skeleton is shown in purple. (b) The algorithm samples initial connected components (blue) in regions (green) around each skeleton vertex. (c) Sampling regions expand outward along the skeleton edges. We depict the regions in the location where samples were generated for clarity; in the actual algorithm the regions advance past the newly generated samples. (d) The components in the middle tunnels successfully connect to form bridges, and their regions are released. The outer passages are still expanding.

rate still gives a strong preference for avoiding regions which are stuck at impassible regions of workspace.

3.4 Dynamic Region sampling for Graph-Based Planners

Dynamic Region RRT provides a fast feasibility planner which addresses single-query settings well. For multi-query settings, a planner should construct a roadmap with good coverage and connectivity of $\mathcal{C}_{free}$ so that diverse queries can be answered efficiently. This is typically best achieved by graph-based planners that build roadmaps with multiple connected components. To tackle problems in this setting, we show how the dynamic region sampling technique can be generalized to account for the separate evolution of multiple connected components in the roadmap. We use the PRM method as the axis for presentation, but the concepts can be employed with any sampling-based planner that can be cast in terms of expanding and connecting components of the roadmap.

The high-level concept for PRM with dynamic region sampling is to view the workspace skeleton as a rough map of the important regions of $\mathcal{C}_{free}$ that we must cover. The skeleton edges describe simple contiguous volumes such as rooms or tunnels, while the skeleton vertices describe

junctions of such volumes. We will refer to the volume of workspace described by a skeleton edge as an *edge segment*, which reflects the concept of a skeleton-induced segmentation of the workspace [50].

Our goal will be to cover each edge segment with a set of vertices that connect the roadmap from the region near the source vertex to the region near the target vertex. We describe a roadmap with this property as *locally connected*. Such a roadmap has good coverage of all distinct regions of workspace, and should be able to quickly answer a wide variety of queries with a path of reasonable cost.

Algorithm 2 Roadmap construction with Dynamic Region PRM

Require: Skeleton $S = \{S_V, S_E\}$, Roadmap $G = \{G_V, G_E\}$

function BUILDROADMAP( )
  Initialize components at each skeleton vertex
  for all $v \in S_V$
    $r \leftarrow$ GETREGIONRADIUS(v.point)
    $Q \leftarrow$ SAMPLEVALIDCONFIGURATIONS($\beta_r(v)$)
    $E \leftarrow \varnothing$
    for all $q \in Q$
      $N \leftarrow$ NEARESTNEIGHBORS(q, Q)
      $E \leftarrow E \cup$ ATTEMPTCONNECTIONS(q, N)
    for all Connected Componet $cc \in \{Q, E\}$
      INITIALIZEREGIONS(cc.vertices, v)
    $G_V \leftarrow G_V \cup Q$
    $G_E \leftarrow G_E \cup E$
  Main PRM Loop
  while $\neg done$ ▷ either node limit or S covered
    $r \leftarrow$ SELECTREGION()
    if $R \neq \varnothing$
      $Q \leftarrow$ EXPANDLOCALCOMPONENT(r)
      while $\exists q \in Q \mid r$.CONTAINS(q)
        ADVANCEREGION(r, Q)
      CONNECTLOCALCOMPONENTS(r.edge)
    else
      $e \leftarrow$ RANDOMUNCONNECTEDSEGMENT( )
      CONNECTLOCALCOMPONENTS(e)

To aid in describing the method, we define a *local connected component* for an edge segment as a set of roadmap vertices which are mutually connected without considering vertices outside of the segment volume. For any two vertices v_a, v_b in a local connected component C, there must be a path from v_a to v_b through some set of vertices $V \subset C$. This concept describes a portion of a roadmap that is locally connected within a particular volume of workspace. A local connected component with vertices near both the source and target of the corresponding skeleton edge will be termed a *bridge*. Bridges represent a connected path that traverses the edge segment volume.

The gist of the method is to generate local connected components near skeleton vertices and extend them across their edge segments with dynamic sampling regions. Local connected components form bridges by either extending all the way across their edge segment or by merging with a local component inbound from the opposite direction.

We begin by initializing sampling regions at each skeleton vertex $v \in S_V$ and sampling a number of configurations within (Alg. 2). Next, we attempt to form connections within each group of samples to form one or more connected components at each skeleton vertex. For each such component C, we initialize a sampling region on each outbound edge e from v and track each tuple (C, e.source, e.target) as our local connected components. This seeds the roadmap with at least a pair of local connected components for each edge e, with an equal number rooted on either end (Fig. 3.5(b)). Note that vertices sampled near a skeleton vertex will be present in more than one local component because they are partially responsible for covering each adjacent skeleton edge segment. We initialize sampling regions for each local connected component on the first point in the edge path it is traversing to lead their extension through the appropriate edge segment.

The sampling regions are used to perform a guided expansion of the local connected components they lead. On each iteration of the algorithm, we select a sampling region r and generate one or more configurations Q within its boundary. We then attempt to connect each valid configuration $q \in Q$ to its nearest neighbors in the local connected component C that r is expanding: on failure, q is discarded. Successful connections are retained and added to C (Fig. 3.5(c), Alg. 3). We then advance r along its skeleton edge path until it no longer touches any of the newly added

Algorithm 3 Component expansion and connection

Require: Roadmap $G = \{G_V, G_E\}$

Expand a local component
function EXPANDLOCALCOMPONENT(Region r)
    $C_r \leftarrow$ GETLOCALCOMPONENT(r)
    $Q \leftarrow$ SAMPLEVALIDCONFIGURATIONS(r)
    for all $q \in Q$
        $N \leftarrow$ NEARESTNEIGHBORS(q, C_r)
        $E \leftarrow$ ATTEMPTCONNECTIONS(q, N)
        if $E = \varnothing$
            $Q \leftarrow Q \setminus q$
            continue ▷ couldn't connect
        $G_V \leftarrow G_V \cup q$
        $G_E \leftarrow G_E \cup E$
    r.UPDATESUCCESSRATE($|Q|$, K)
    return Q
Connect local components in a segment
function CONNECTLOCALCOMPONENTS(SkeletonEdge e)
    Draw samples at a random point on the edge
    $p \leftarrow$ RANDOMPOINT(e.path)
    $r \leftarrow$ GETREGIONRADIUS(p)
    $Q \leftarrow$ SAMPLEVALIDCONFIGURATIONS($\beta_r(p)$)
    Attempt to merge components
    $C \leftarrow$ GETLOCALCOMPONENTS(e)
    for all $q \in Q$
        $E \leftarrow$ ATTEMPTCONNECTIONS(q, C)
        if E has edges to more than one $c \in C$
            merge all $c \in C$ connected by E

samples (Alg. 4). In this way, the sampling region r tracks the component C's progress in covering the edge segment.

If r successfully expands C, we additionally attempt to connect the retained samples in Q to any local components inbound on this edge segment from the opposite direction. This is to make the algorithm aggressively attempt to form bridges at the earliest opportunity. On forming a bridge, we merge the newly connected local components and release their sampling regions, which are no longer needed (Fig. 3.5(d)).

If r advances to the end of its edge without C connecting to a local component rooted at the

target skeleton vertex s_t, then C has formed a bridge but not yet connected to the roadmap locally near s_t. In this case, we generate local components with the new vertices Q on the edges outbound from s_t to continue searching for a connection to the local components already rooted at s_t. This ensures that the algorithm continues to explore until the roadmap is locally connected or disjoint global connected components cover the skeleton. The latter can happen in problems with disjoint regions of $\mathcal{C}_{free}$ (Fig. 3.6(b)).

To ensure that disjoint local components within an edge segment are connected, we apply an additional connection stage after each expansion step. We pick a random point p along r's skeleton edge path and sample a set of valid configurations Q. For each $q \in Q$, we attempt to form connections between at least two local connected components within this edge segment. Any configurations that form the necessary connections will be added to the roadmap and will trigger a merge of the corresponding local connected components (Alg. 3).

As in the expansion step, a merge of two components coming from opposite directions forms a bridge and releases their sampling regions. Similarly, a merge with an existing bridge absorbs all affected vertices into the bridge. When two components from the same side merge, we retain the sampling region which has advanced the farthest along the edge path.

The set of initial regions will expand their corresponding components outward from their root skeleton vertex in a similar fashion as in Dynamic Region RRT possibly extending to the end of the skeleton. When selecting a region to expand on each iteration, we can employ a weighted random choice to favor regions which have been more successful in expanding the roadmap. The weight for each region is calculated by the success rate of extending into that region. To ensure that region weights represent the recent history of performance, we can apply a discount factor between zero and one to the prior weight before updates. A weighting based on success rate ensures that the algorithm will explore the edge segments which have proved to be traversable before expending effort on segments which are difficult to connect or even not path-connected in $\mathcal{C}_{free}$.

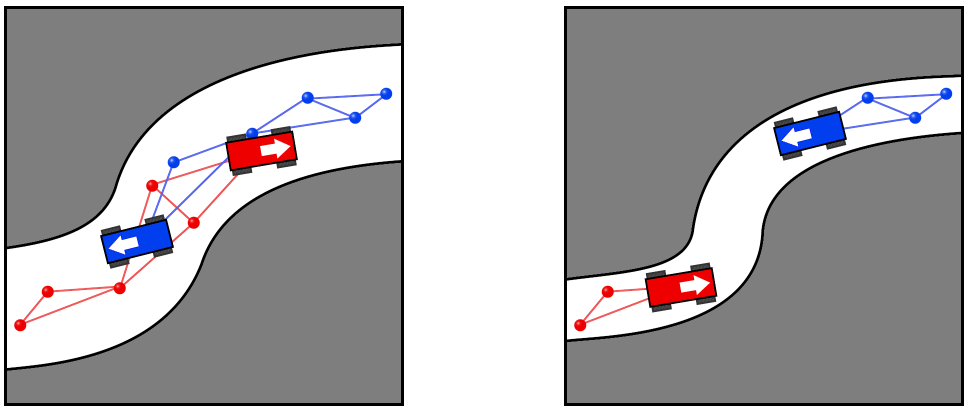
Algorithm 4 Dynamic Region operations

```
1: ***Initialize regions and local components***
2: function INITIALIZEREGIONS(Configurations Q, SkeletonVertex v)
3:     for all e ∈ v.GETOUTBOUNDEDGES( )
4:         C ← MAKELOCALCOMPONENT(e, v, Q)
5:         CREATEREGION(C)
6: ***Advance an expansion region one step***
7: function ADVANCEREGION(Region r, Configurations Q)
8:     ***Check for end of edge***
9:     p ← r.GETNEXTSKELETONEDGEPOINT( )
10:    if ∄p
11:        ***Attempt to merge components***
12:        C ← GETLOCALCOMPONENTS(r.edge.target)
13:        for all q ∈ Q
14:            E ← ATTEMPTCONNECTIONS(q, C)
15:            if E ≠ ∅
16:                merge all c ∈ C connected by E
17:        if not merged
18:            INITIALIZEREGIONS(Q, r.edge.target)
19:        DELETEREGION(r)
20:    else
21:        ***Move to the next position***
22:        r.center ← p
23:        r.radius ← GETREGIONRADIUS(p)
```

3.4.1 Local Connectivity

A straight-forward application of the dynamic region sampling paradigm for RRT methods is very likely to produce disconnected roadmaps because there is no mechanism ensuring that the samples produced within a region r will connect to other vertices within r's edge segment. This is implied in RRT methods due to tree extension, but not guaranteed for PRM methods which form local plans rather than growing towards a new sample. This motivates our choice of requiring sampling regions to expand a particular local connected component.

Even with this consideration, it is quite possible that local components may grow past each other along a skeleton edge and fail to connect when a connection is feasible (Fig. 3.6(a)). This motivates the need for a separate connection step to provide a guidance mechanism for completing



(a) (b)

Figure 3.6: Two examples of connection problems that can occur with disjoint connected components. The robot is a car-like vehicle with mecanum wheels, with an arrow indicating its orientation. Two connected components of the roadmap are shown in red and blue. (a) An example of a missed connection. The components are connectable because the mecanum wheels permit the robot to turn in place. (b) An example of a locally disjoint $\mathcal{C}_{free}$There is no way to merge these components within the tunnel because the car can't turn in the available space.

the connections to achieve local connectivity.

However, we note that a locally connected roadmap does not necessarily express a complete coverage of $\mathcal{C}_{free}$ because $\mathcal{C}_{free}$ can have locally disjoint components within a particular edge segment. Consider an example where a car-like robot must traverse a tunnel that is too narrow to turn around (Fig. 3.6(b)). Within the tunnel, there are two disjoint regions of $\mathcal{C}_{free}$: one for each direction of travel. In more complex examples with three-dimensional environments or mobile manipulators, there could be many more locally disjoint components that are only connected in some specific areas of the environment. The algorithm attempts to account for this by creating new local components when a sampling region completes a skeleton edge without connecting to the other side. This encourages construction of a roadmap that presents some level of coverage for locally disjoint regions of $\mathcal{C}_{free}$.

3.4.2 Answering Queries

When presented with a query consisting of a start configuration $q_{start} \in \mathcal{C}_{free}$ and a goal region $Q_{goal} \subset \mathcal{C}_{free}$, we sample a $q_{goal} \in Q$ and attempt to connect q_{start}, q_{goal} to the roadmap. If the roadmap has adequately covered $\mathcal{C}_{free}$, this should be straight-forward.

It is possible that either start or goal q is not connectable to the current roadmap. This represents a case where either the skeleton missed the corresponding parts of workspace (resulting in no configurations nearby) or the nearby configurations lie in a region of $\mathcal{C}_{free}$ that is locally disconnected from q. The repair strategy is to expand rapidly outward from q in search of either the roadmap (thus completing the connection) or the skeleton (thus allowing the use of dynamic region guidance to complete the connection). An RRT is ideal for this purpose as it handles both cases elegantly: it will rapidly find a nearby skeleton point, either leading to a connection or arrival at a region near a skeleton vertex where dynamic region guidance can be employed. This is analogous to the Spark PRM strategy [51] where an RRT is locally employed to bridge narrow passages for a PRM planner.

3.4.3 Validation

To evaluate Dynamic Region sampling with PRM, we tested the method on two problems with multiple path homotopy classes and compared against PRM (baseline), PRM with Workspace Importance Sampling (WIS) [5], and Dynamic Region RRT(DR-RRT).

The environments include a `Garage` problem with a quadcopter robot, a `DhaA` protein with a ligand probe, and a cramped three-dimensional `GridMaze`. Each environment exhibits winding tunnels which increase the difficulty of connecting configurations. In each problem, the PRM planners build an initial roadmap with a fixed number of vertices before being presented with a series of queries. They then search for a solution for each query in sequence, starting from the current roadmap and expanding it if necessary. This exercises the multi-query intention of PRM and shows how well the constructed roadmaps generalize over several planning requests. The Dynamic Region RRT method is included to contrast the performance of a guided single-query

method.

3.4.3.1 Experiment Setup

All methods were implemented in a C++ motion planning library developed in the Parasol Lab at Texas A&M University. All experiments were executed on a desktop computer running CentOS 7 with an Intel® Core™ i7-3770 CPU at 3.4 GHz, 16 GB of RAM, and the GNU g++ compiler version 4.8.5. Skeletonization for the dynamic region methods was performed with a Mean Curvature Skeleton [40] implemented in the CGAL library [52]. Workspace tetrahedralization for WIS was performed with a combination of the TetGen [37] and CGAL libraries. Time to build these models was considered pre-processing and not included in the result plots.

Each experiment ran until all queries solved. We performed 35 trials for each experiment and report the initial roadmap construction time, time to solve each query, and cost of the produced paths. Construction and Query time are reported in seconds, while path cost is in euclidean distance in $\mathcal{C}_{space}$. Each trial is plotted as a scatter dot to illustrate the spread of behavior, with jitter to clarify overlaps.

PRM and WIS sample ten configurations per iteration in all environment. Dynamic Region PRM uses five in `Garage` and `Gridmaze` because it generates all samples for an iteration within the same locality and additionally attempts a second set of samples during the connection phase. In `DhaA` it uses ten samples to reflect the greater difficulty of sampling a valid configuration for the ligand probe. All PRM methods use eight nearest-neighbors. Dynamic Region PRM employs clearance-based sampling, while Dynamic Region RRT uses fixed-size regions to keep the tree compact near the skeleton.

3.4.3.2 Analysis

The `Garage` problem (Fig. 3.7) presents a series of ten queries scattered across the levels of the structure. The space is relatively open compared to the robot size, and the primary sources of difficulty are thin walls and large scale. We observe that Dynamic Region PRM consistently takes longer to construct an initial map than PRM or WIS, and struggles slightly on the second query.

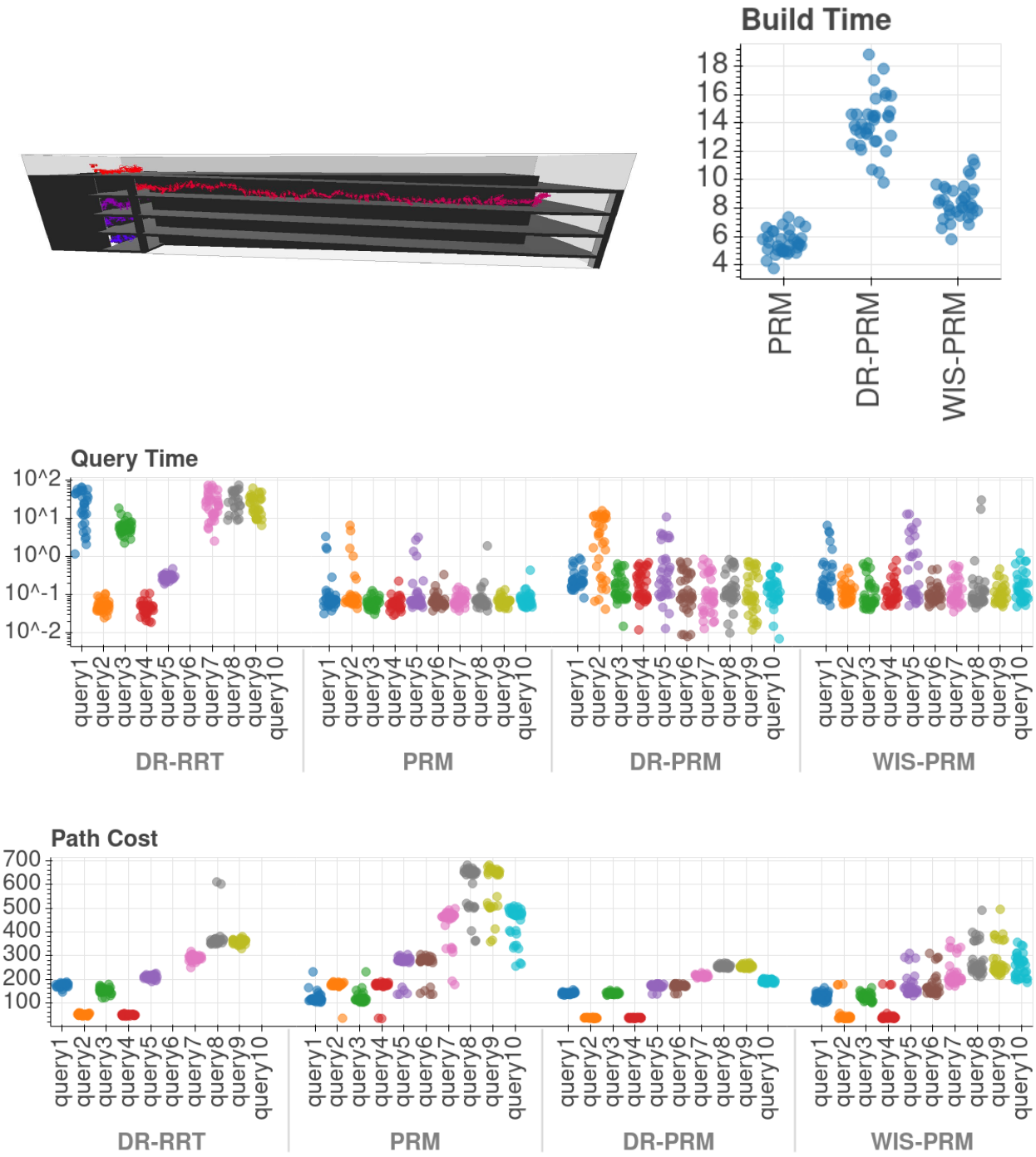


Figure 3.7: Evaluation of Dynamic Region PRM in the `Garage` environment. DR-RRT failed to solve queries six and ten at all within an 80 second time limit, and occasionally failed queries one (four fails), eight (ten fails), and nine (three fails).

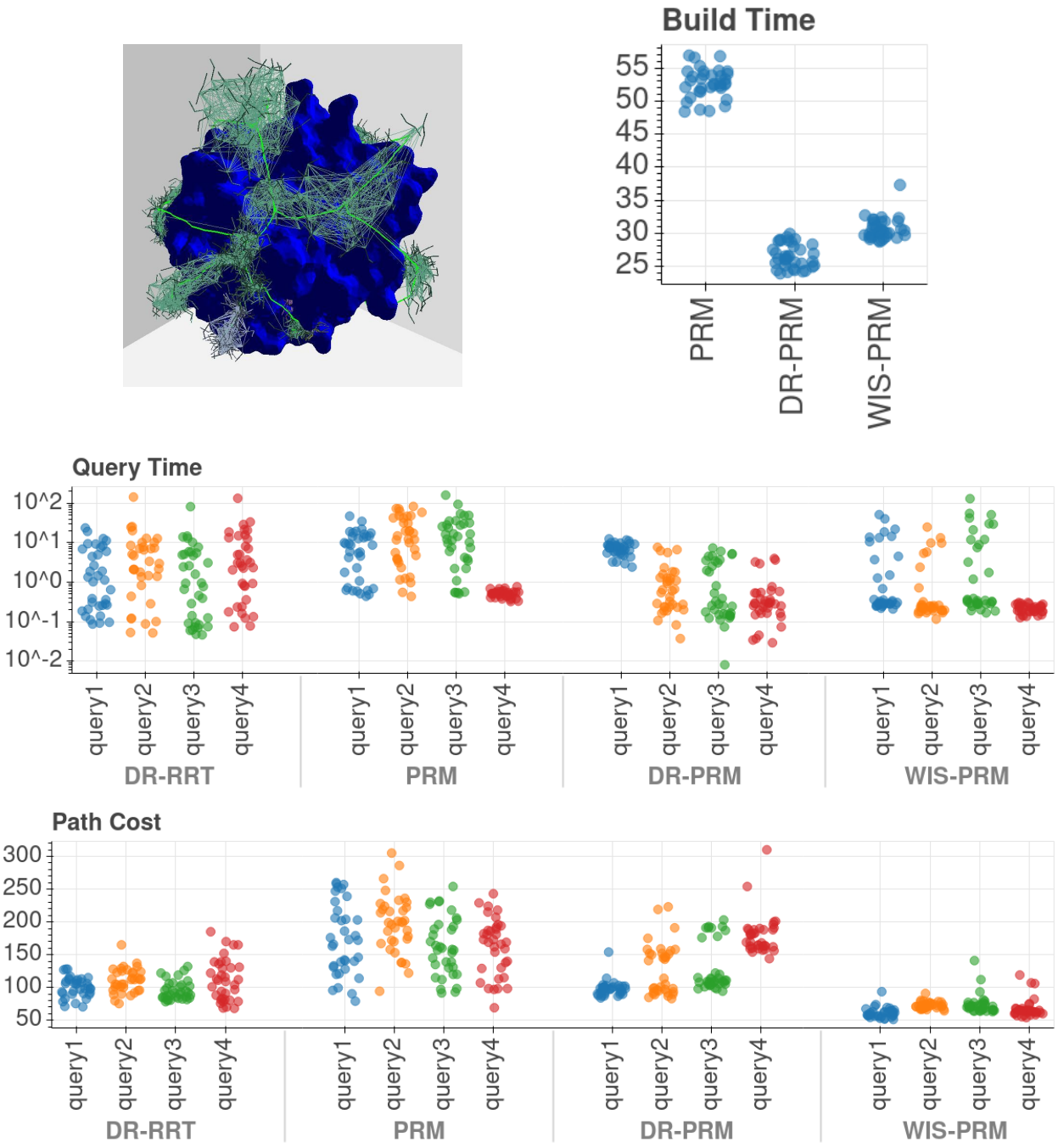


Figure 3.8: Evaluation of Dynamic Region PRM in the `DhaA` environment.

However, it consistently produces very low path costs with little variance. This occurs because the region guidance forces the planner to cover a regular volume around the skeleton edges, which provides a roadmap with paths that roughly map to paths through the skeleton (plus any distance

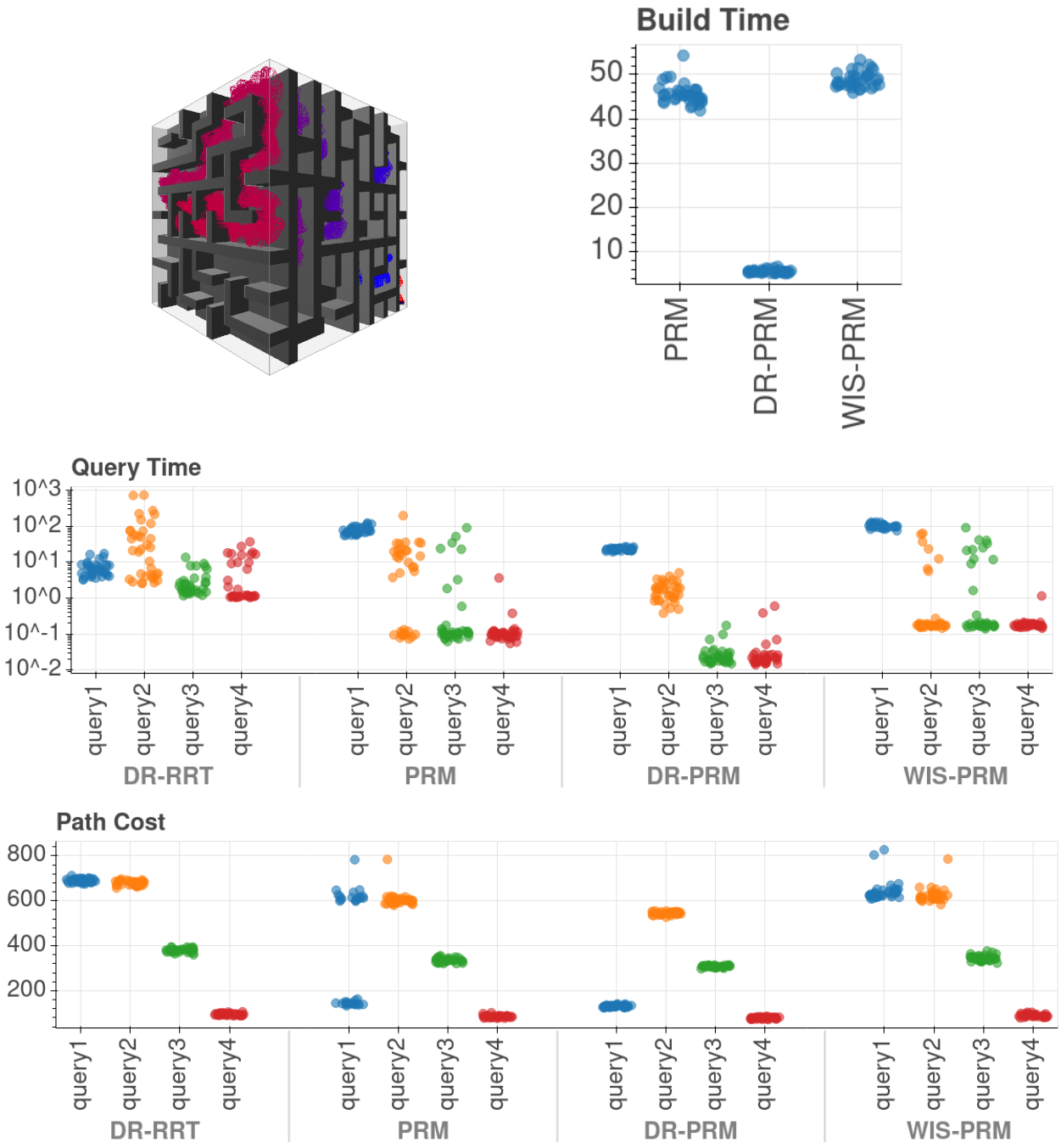


Figure 3.9: Evaluation of Dynamic Region PRM in the `Gridmaze` environment.

needed to reach the skeleton if the query is far away). WIS produces better paths than PRM by taking greater care to sample in less accessible regions of workspace, thus providing coverage that

is better but not as consistent as Dynamic Region PRM. Dynamic Region RRT can sometimes match its PRM counterpart's path cost, but always takes longer to do so and frequently fails to find a path within a reasonable time limit (80 seconds here).

The `DhaA` problem (Fig. 3.8) presents a sequence of four queries representing ligand binding sites. The first three queries have start and goal positions close to the skeleton, while the fourth is farther away. We see that Dynamic Region PRM has the fastest build time, although the advantage is not highly significant over WIS. Its query time however is consistently low, whereas the other methods exhibit a significant spread of times. Path cost is better than PRM but not as low as WIS; this occurs because the Dynamic Region paths follow the skeleton closely, while the WIS paths hug the boundary relatively closely. The path for the fourth query is longer for Dynamic Region PRM because it lies farther from the skeleton. Here the nearest nodes are concentrated around the skeleton, so the path effectively 'snaps' to the skeleton's topology. This case illustrates a possible negative side-effect of skeleton guidance. Dynamic Region RRT exhibits a similar issue with lower intensity due to constant-sized regions. However, its query time is subject to long-running outliers when the algorithm gets stuck trying to break through a low-clearance area.

The `Gridmaze` problem (Fig. 3.9) presents a sequence of four queries dispersed in the maze. The maze is fairly tight, making the entire workspace relatively close to the skeleton. In this setting Dynamic Region PRM excels with rapid build and query times compared with the other methods. Its path cost is also consistent and minimal, which is expected given the close matching between the workspace and its skeleton. This is an ideal case for Dynamic Region PRM, even over its RRT counterpart which fails to discover the cheapest path for the first query.

3.5 Summary

Dynamic Region sampling is a general paradigm that can be applied to any sampling-based planner to focus its exploration along the salient paths through workspace, just ahead of the tree frontier. It supports both fast feasibility planning with RRT and building well-connected roadmaps with PRM. While not applicable to all planning problems, our characterization shows that it works well for many problems requiring locomotion of a physical robot.

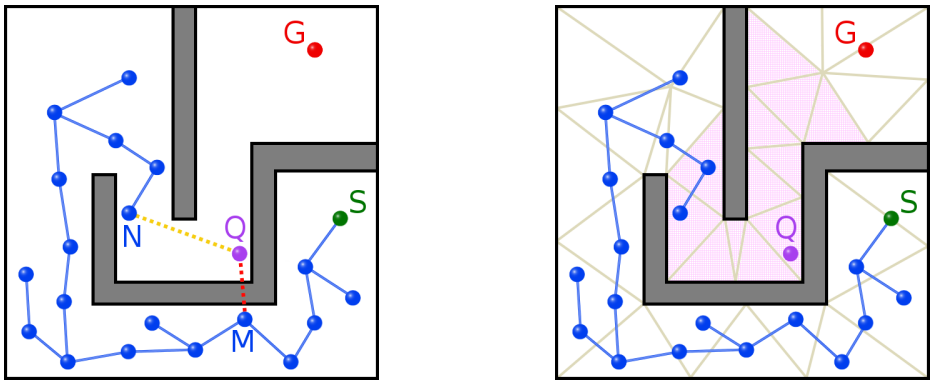
4. TOPOLOGICAL NEAREST-NEIGHBOR FILTERING*

Sampling-based motion planners such as PRM [11] and RRT [15] explore a problem by sampling random configurations for a robot and connecting them to nearby neighbors. To determine which configurations to connect, planners use a nearest-neighbor algorithm to determine which existing roadmap configuration q_{near} is closest to a new sample q_{new}. In this context, the nearness of two configurations is determined by a distance metric defined over the entire configuration space. This is problematic because the topology of $\mathcal{C}_{free}$ is very different from that of $\mathcal{C}_{space}$ where the distance is measured. As a result, standard nearest-neighbor algorithms produce very poor choices in the presence of thin walls, which we denote as the *thin wall* problem (Fig. 4.1(a)).

As indicated in Chapter 2.2, nearest-neighbor finding is a major bottleneck for sampling-based planners. However, prior work has focused primarily on utilizing faster computational techniques as opposed to leveraging the shape of the free space.

Topological Nearest-Neighbor Filtering examines how a model of workspace connectivity can be employed to identify a reduced set of candidate neighbors which are nearby through connected $\mathcal{C}_{free}$ [10]. Proximity through connected workspace is an important factor in estimating the probability that a connection will succeed because workspace is a dilated model of the translational subspace of $\mathcal{C}_{free}$ for all physical robots with translational degrees of freedom (Fig. 4.1(b)). A path through $\mathcal{C}_{free}$ necessarily implies a corresponding path through workspace, even though the converse isn't necessarily true. Identifying the candidate neighbors for which a workspace path exists thus identifies a superset of the connectable neighbors which meet at least one criterion for being connectable.

*Portions of this chapter describing the core concepts and application to k-nearest neighbors are reprinted with permission from [10] R. Sandström, A. Bregger, B. Smith, S. Thomas, and N. M. Amato. "Topological nearest-neighbor filtering for sampling-based planners," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 3053–3060, 2018. © 2018 IEEE.



(a) Problem with pure proximity (b) Topological Filtering

Figure 4.1: A sketch of the Topological Filtering concept. (a) An example scenario where proximity alone is not a good metric for choosing a nearest-neighbor. The start S and goal G are shown in green and red, and the roadmap is shown in blue. The planner is searching for a nearest-neighbor for the sample Q. The best neighbor is N, but nearest-neighbor algorithms based purely on a proximity metric will choose M because it is closer in $\mathcal{C}_{space}$. (b) Topological filtering identifies a set of candidate cells (purple) which are nearby to the cell containing Q. Configurations within are passed to the nearest-neighbor algorithm as the input set. Reprinted with permission from [10], © 2018 IEEE.

4.1 Method

Topological Filtering relies on a convex cell decomposition of the workspace, which is a partitioning of the free workspace into a set of discrete convex cells [36]. The decomposition provides a graph-representation of adjacent convex cells in workspace and can be thought of as a coarse atlas of the space with the cells as charts. This map encodes information about the connectivity of the cells and can be searched to locate sets of nearby cells, or *neighborhoods*.

The decomposition graph can also be used as a means of 'bucketing' nearby configurations together. A point on the robot's base, termed the *reference point*, is chosen to represent the robot's rough location in workspace. Usually the object's centroid or bounding box center is a good choice for this point. Let r be a robot and p be its reference point. For any free configuration q of r, a cell c in the decomposition graph W is said to *contain* q if $p \in c$ when r is configured at q (as in [7], Fig. 4.10(a)). The cell c which contains p may be efficiently determined by a range-searching method such as a segment tree [36] or spatial hash [53] to a grid with a precomputed mapping

between voxels and colliding decomposition cells. Since the cells of W are disjoint, each free configuration will map to exactly one cell (boundary cases may be decided by any deterministic method).

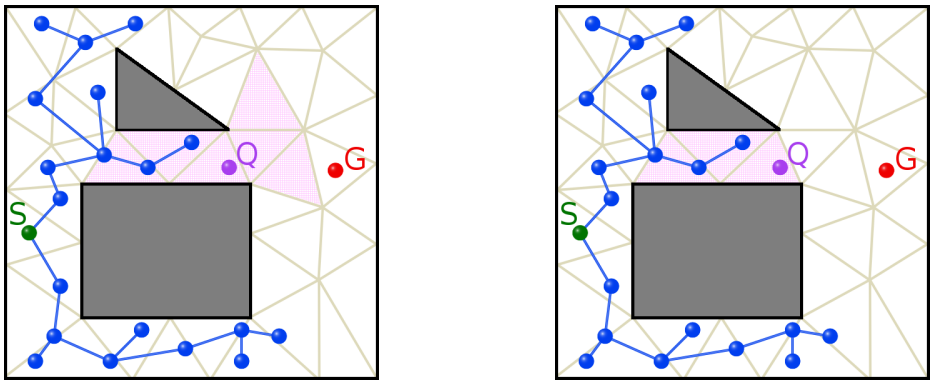
Algorithm 5 Topological Filtering

```
1: ***Find topological candidates for a query configuration q w.r.t. a topological map m.***
2: function FINDTOPOLOGICALCANDIDATES(Configuration q, TopologicalMap m)
3:     cell ← m.FINDCONTAININGCELL(q)
4:     F ←FINDTOPOLOGICALFRONTIER(cell)
5:     candidateCfgs ← { }
6:     for all cell ∈ F
7:         cfgs ← m.GETCONTAINEDCFGS(cell)
8:         candidateCfgs ← candidateCfgs ∪ cfgs
9:     return candidateCfgs
```

The topological filter leverages this relationship by mapping configurations to their containing cells with a hash map. Whenever a configuration q is added to the roadmap, the filter maps q to the set for cell c, and conversely maps c to q. This *topological mapping* provides an amortized constant-time lookup of the vertices in a cell and the cell holding an already-discovered vertex (although the latter can be efficiently recomputed as noted above).

When searching for nearest-neighbors to some configuration q, the filter first locates the cell c which contains q. Next, it performs a single-source shortest path search through the decomposition graph W starting from q to identify a set of cells $F \subset W$ called the *topological frontier* for this query. F is a set of cells which hold the most promising nodes for connection w.r.t. the subset of $\mathcal{C}_{space}$ that W is modeling (in this case, the physical workspace). The specific manner of selecting F will depend on the type of nearest-neighbor process we wish to achieve (k-nearest or radius-based).

The roadmap vertices in F can be determined using the topological map, which we call the *topological candidates* for the query configuration q (Alg. 5). These are passed to a standard



(a) Standard Frontier (b) Query-Relevance Frontier

Figure 4.2: An example of Topological Frontiers for k-NN. The candidate cells are shaded pink. The start S is shown in green and the goal G in red. The workspace decomposition is shown in beige, and the current roadmap is shown in blue. For illustration purposes, we use hops as the distance metric and set $\delta_{threshold} = 1$. (a) The query point Q is one hop from the nearest populated cell, so the candidate cells are the set of cells within two hops of Q. (b) The candidate cells are reduced when observing query relevance by ignoring edges that lead closer to the goal. Reprinted with permission from [10], © 2018 IEEE.

nearest-neighbor method as the input candidates. As such, the filter's role is to efficiently select a small set of promising nearest-neighbors for some nearest-neighbor algorithm.

4.2 Filtering for k Nearest-Neighbors

For k-nearest neighbors, the search for the topological frontier F begins from the initial cell c containing the query point and looks for the first cell c_1 that holds a mapped configuration. It then continues searching until exceeding an additional *backtrack distance* $\delta_{threshold}$, and takes all cells discovered since (and including) c_1 (Alg. 6, Fig. 4.2(a)). This frontier represents a dynamically-selected set of cells that include the most topologically relevant configurations for the query point. The backtrack distance serves as a smoothing of the approximate relationship between workspace and $\mathcal{C}_{free}$ to prevent over-fitting of the candidate set based on an imperfect approximation.

The frontier F may be very large and far from the original cell c if c is far from the covered free space, and would seem to be not topologically relevant to c in that case. However, there is no set of candidate cells with better topological relevance, and the set of configurations in F will

typically be much smaller than the entire roadmap.

This frontier may be empty if there are no roadmap configurations in any cell that is connected to c - in this case, there are no connectable configurations in the roadmap. A non-empty frontier is guaranteed to hold at least one configuration by construction, which will be relatively nearby through connected workspace compared to other configurations in the roadmap.

The worst case complexity for locating this frontier occurs when there are no configurations in the roadmap. In this case, the search executes a complete single-source shortest paths algorithm over the entire set of cells reachable from c in the decomposition graph. This may be expensive initially, but as the roadmap coverage of free space increases, it is more and more likely that F will represent a topologically relevant neighborhood. The graph search can also be cached so that the frontier can be recomputed in linear time in the number of discovered cells on each reuse. As the frontier will shrink over time, the filter's efficiency and efficacy improve as the roadmap grows in size.

Nonetheless, it is pertinent to choose a fairly coarse decomposition to minimize the cost of the candidate cell search. Extremely fine decompositions provide more stratification of neighborhoods than is needed for the filter, and offer little benefit in return for the additional overhead. A good heuristic is to choose a convex decomposition that generates the smallest number of well-formed cells needed to cover the environment.

4.2.1 Query Relevance Variant

For tree-based methods, choosing neighbors that are relevant to solving the query is equally important because performance depends on choosing cells that are likely to generate productive extensions from an 'earlier' region of the problem (i.e., closer to the query start) towards a 'later' region (i.e., closer to the query goal). We refer to candidate cells meeting this criteria as *query relevant*.

The filter can be adjusted to consider the query relevance of potential candidate cells by determining a subset of the original adjacency map for the decomposition graph which discards edges leading away from the goal. This can be computed as a pre-processing step with a single-source

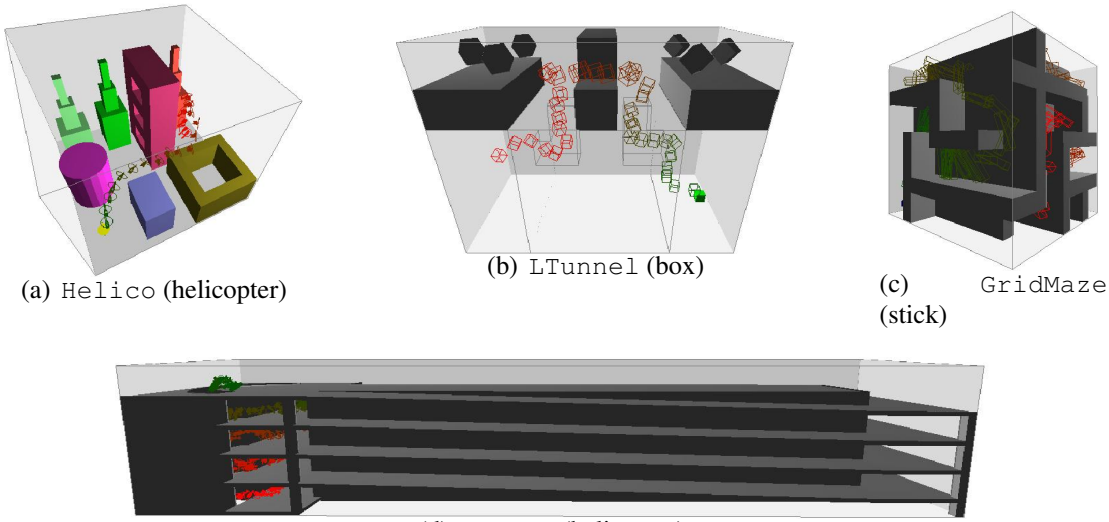
Algorithm 6 Frontier selection for k-nearest neighbors.

```
***Topological relevance only.***
function FINDTOPOLOGICALFRONTIER(Cell cell)
    map ← Original adjacency map
    return FINDTOPOLOGICALFRONTIER(cell, map)

***Topological and query relevance.***
function FINDQUERYSAMPLINGFRONTIER(Cell cell)
    ***Initialize a query-relevant adjacency map.***
    if map is empty or goal has changed
        map_o ← Original adjacency map
        map ← SSSP(goal, map_o)
        ***Add cross edges.***
        for all v ∈ decomposition.vertices
            for all adj ∈ v.neighbors
                if map[v].score < map[adj].score
                    map[v].successors ← {adj} ∪ map[v].successors
    return FINDTOPOLOGICALFRONTIER(cell, map)

***General frontier finding.***
function FINDTOPOLOGICALFRONTIER(Cell cell, AdjacencyMap m)
    stop ← visited a cell with distance at least δ_threshold
               greater than the first populated cell
    childMap ← SSSP(cell, m, stop)
    ***Pick out the frontier.***
    frontier ← { }
    for all parent ∈ childMap.keys
        if ISPOPULATED(parent)
            frontier ← frontier ∪ {parent}
    return frontier

***Run a SSSP algorithm on a decomposition graph d from cell c, and terminate when t is true.***
function SSSP(Cell c, DecompositionGraph d, TerminationCritera t)
    return set of discovered cells and distances from c
```



(a) `Helico` (helicopter)

(b) `LTunnel` (box)

(c) `GridMaze` (stick)

(d) `Garage` (helicopter)

Figure 4.3: Test environments for Topological Filtering with k-NN. Selected paths from RRT with filtering are shown from start (green) to goal (red). Reprinted with permission from [10], © 2018 IEEE.

shortest paths algorithm starting from the cell which contains the query goal. The resulting distances are a measure of each cell's proximity to the goal through connected workspace: smaller distances indicate closer proximity. Adding the cross-edges creates an adjacency map where each cell's successors are of equal or greater distance from the query (Alg. 6, Fig. 4.2(b)).

The filter can use this mapping instead of the original when exploring for the sampling frontier. This further limits the frontier to those cells that are 'behind' the original cell c relative to the goal to encourage productive extensions. The method could feasibly be applied with other types of adjacency maps for specialized problems.

4.2.2 Validation

We evaluate the method by comparing nearest-neighbor and planning times for several RRT methods with and without the filter and query relevance. The planners used include RRT [15], SST [23], Dynamic Region RRT [9] (DR-RRT), SyCLoP [7], and a modified version of SyCLoP for holonomic problems referred to as SyCLoP-holo. These were selected as a spectrum of different levels of heuristic guidance. The unfiltered methods use a brute-force nearest-neighbor search

to isolate the gains produced by the filter.

The modified SyCLoP-holo algorithm is a modification of SyCLoP for holonomic problems, in which there are no controls to sample. This variant is identical to the original except that (a) a sampled configuration q_{rand} is used as the growth target as in standard RRT, and (b) once a workspace cell is selected from the discrete lead, we use a traditional neighborhood finder to select q_{near} from the set of configurations within (rather than using the selection history as in the original method). These changes aim to adapt SyCLoP to holonomic problems while retaining the spirit of the method. Since SyCLoP and SyCLoP-holo are already choosing q_{near} from a single decomposition cell, it does not make sense to subsequently apply our filter. We have included them for comparison because they also employ a decomposition to locate configurations for extension.

4.2.2.1 Experiment Setup

We use four simulated environments for the evaluation with different aspects of interest (Fig. 4.3). `Helico` is relatively open. `LTunnel` presents three narrow entrances. `Garage` has a four-story winding ramp and additional longer routes toward the other end of the environment. The `GridMaze` environment has long, winding paths in a cramped tunnel that constrains the robot's rotational DOFs. Maze-like problems are notoriously difficult for RRT's because the sampled target configurations q_{rand} are quite frequently located across the maze walls, resulting in short, erratic extensions that scrape very close to the obstacle space. Additionally, corners and tight turns frequently create portions of $\mathcal{C}_{space}$ where only a very small portion of the local sampling volume can yield a configuration that extends the tree around the corner.

Nonholonomic trials were performed in the `Helico` and `LTunnel` environments. Nonholonomic robots increase the problem difficulty by increasing the dimensions of the planning space and severely limiting the allowed actions to the robot's control set.

The robots in all cases are six DOF rigid bodies. The RRT maximum extension distance is set to approximately the bounding sphere radius for the robot, and a tetrahedralization is used for the workspace decomposition. The nonholonomic robots are fully actuated with simple discrete control sets (i.e., over a single extension the robot can exert a force on itself in any one of its

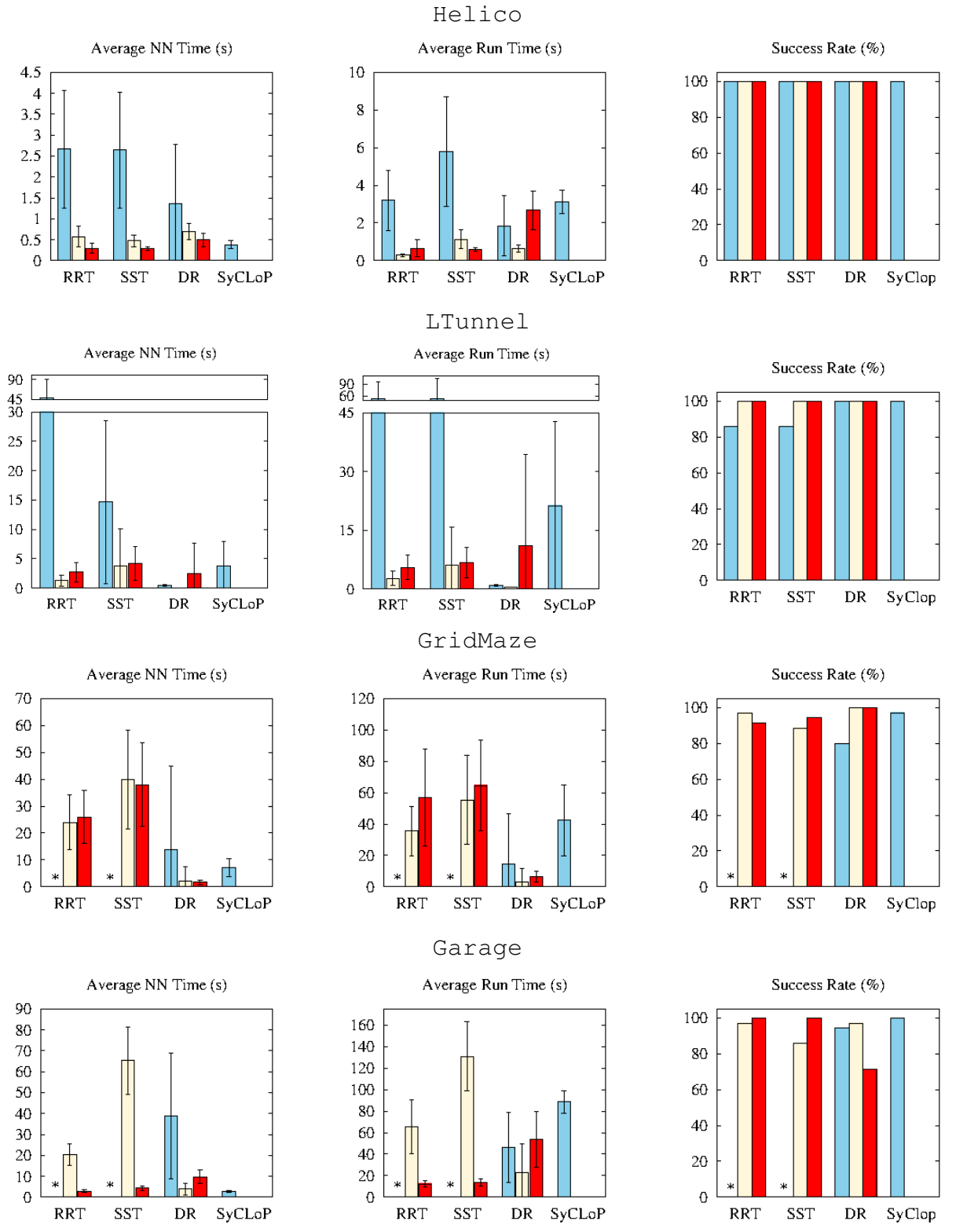


Figure 4.4: Evaluation of the topological filter with k-NN in holonomic problems. Error bars indicate standard deviation. A * indicates that no trials were successful. SyCLoP refers to SyCLoP-holo. Reprinted with permission from [10], © 2018 IEEE.

position or orientation DOFs). An even mix of best and random controls were used for RRT, SST, and DR-RRT, while random controls were used for SyCLoP (as it does not use a growth target).

All experiments were executed on a desktop computer running CentOS 7 with an Intel® Core™ 2 Quad Q9550 CPU at 2.83 GHz, 8 GB of RAM, and the GNU g++ compiler version 4.8.5. The workspace tetrahedralization was performed with a combination of the TetGen [37] and CGAL [52] libraries.

Thirty-five trials are run for each evaluation. Each run is limited to a maximum time of three minutes to complete the queries illustrated in Fig. 4.3. Executions which do not solve the query in this time are considered failures. We report the success rate for each planner and the average execution and nearest-neighbor times for the successful runs (Fig. 4.4). Error bars indicate standard deviation in all cases. Statistical significance is measured with Welch's t-test on the successful trials.

The reported run times do not include the time needed to decompose the workspace; the slowest environment to decompose is the `Garage`, which takes about one second.

4.2.2.2 Analysis

Topological Filter: A consistent drop in nearest-neighbor time is observed for all planners when using the topological filter without query relevance (in both holonomic and nonholonomic trials). The difference from the unfiltered method is highly significant ($p_{\text{val}} \leq .01$) in all cases except DR-RRT in `GridMaze` ($p_{\text{val}} = .05$).

The effect on overall planning time is positive with high confidence ($p_{\text{val}} \leq .01$) in the holonomic trials, excepting DR-RRT in `GridMaze` (improvement with $p_{\text{val}} = .08$). In the other cases, significant improvements are observed in both total time and variance. In the nonholonomic trials, planning time improved with high confidence ($p_{\text{val}} \leq .01$) for SST in both environments and RRT in `LTunnel`. The other three cases showed low confidence improvements.

The unguided planners RRT and SST also attained a unilateral increase in success rate, and are able to reliably solve the holonomic `GridMaze` and `Garage` problems within the three minute time limit. The guided planner DR-RRT sees equivalent or better success rate in all cases.

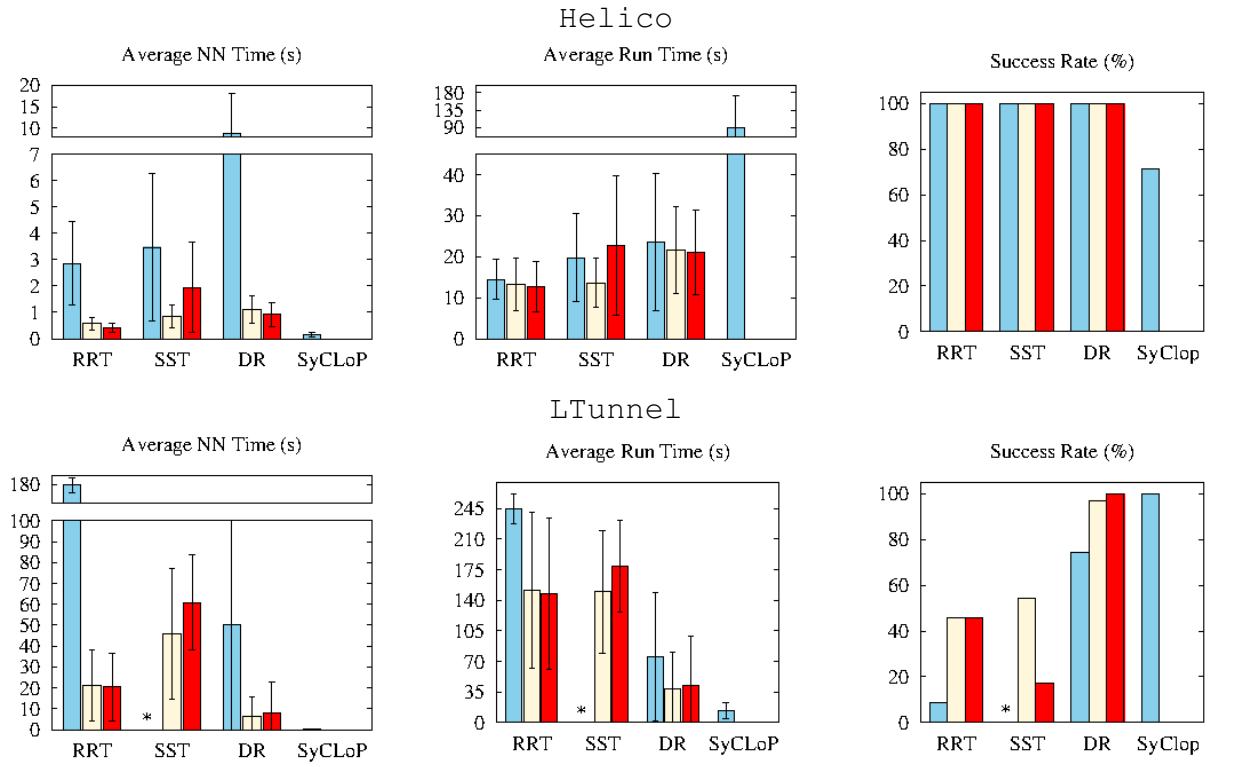


Figure 4.5: Evaluation of the topological filter with k-NN in nonholonomic problems. Error bars indicate standard deviation. A * indicates that no trials were successful. Reprinted with permission from [10], © 2018 IEEE.

Query Relevance: The query relevance option presents mixed results for both nearest-neighbor and planning time. The nearest neighbor time is frequently worse compared with the plain filter. In `Helico`, small differences are seen for all planners with high confidence ($p_{\text{val}} \leq .01$) except RRT ($p_{\text{val}} = .90$) and nonholonomic DR-RRT ($p_{\text{val}} = .12$). In holonomic `LTunnel`, all methods require more nearest-neighbor time than the plain filter with high confidence ($p_{\text{val}} \leq .01$) except SST ($p_{\text{val}} = .77$). The nonholonomic `LTunnel` shows low confidence differences across the board ($p_{\text{val}} \geq .23$). In `GridMaze`, all planners showed low confidence differences ($p_{\text{val}} \geq .43$). In `Garage`, RRT and SST used significantly less nearest-neighbor time with query relevance ($p_{\text{val}} \leq .01$), while DR-RRT showed a high confidence increase ($p_{\text{val}} \leq .01$).

The overall planning time with the query relevance option is frequently worse than the regular filter, with the most extreme cases occurring for the guided planner DR-RRT. The holonomic `Helico` trials were mixed: RRT showed a low confidence degradation ($p_{\text{val}} = .27$), SST showed a high confidence improvement ($p_{\text{val}} \leq .01$), and DR-RRT showed a high confidence degradation ($p_{\text{val}} \leq .01$). The nonholonomic trials exhibited the opposite trend: RRT and DR-RRT improved with low confidence ($p_{\text{val}} \geq .69$) while SST degraded with high confidence ($p_{\text{val}} \leq .01$). In holonomic `LTunnel` we see a unilateral increase in planning time for all planners, with low confidence for SST ($p_{\text{val}} = .73$) and high confidence for RRT and DR-RRT ($p_{\text{val}} \leq .01$). In the nonholonomic version RRT attained a slight improvement but with very low confidence ($p_{\text{val}} = .91$), while the other two planners again degraded with low confidence ($p_{\text{val}} \geq .29$). In `GridMaze`, the planning time increased over the regular filter with high confidence for RRT ($p_{\text{val}} \leq .01$) and low confidence for SST and DR-RRT ($p_{\text{val}} = .2$ and $p_{\text{val}} = .03$, respectively). In `Garage`, RRT and SST saw large improvements while DR-RRT performed worse, with high confidence ($p_{\text{val}} \leq .01$) in all cases.

Success rate with query relevance was roughly equivalent to the regular filter in most cases. Reductions are observed for RRT in `GridMaze`, DR-RRT in `Garage`, and SST in the nonholonomic `LTunnel`.

The SyCLoP planner shows a small amount of nearest-neighbor time because we attempt to

connect configurations which are very near to the goal directly; we found this to necessary in practice to obtain reasonable results. In the nonholonomic `Helico`, SyCLoP wastes some effort evenly expanding the tree; however the `LTunnel` results show that this method is highly effective in less expansive environments.

The nearest-neighbor time for the query relevance variant is generally worse than for the plain topological filter because the more restrictive filter returns no candidates with higher probability, and therefore requires more attempts over the entire execution. For DR-RRT, this restrictiveness appears to interfere with the guidance heuristic in many cases. In these scenarios, the query relevance filters out candidates that were expected by the guidance heuristic. Specifically, DR-RRT employs moving sampling regions from which new samples are drawn; if the sampling regions are moving perpendicular to the flow prescribed by query relevance, this creates contention where the filter could reject the nearby configurations as not en route to the query relative to the sampling region.

For the unguided planners RRT and SST, the filters provide an immense boost in both efficiency and reliability in all cases. The query relevance variant produced good results in `Garage` for these planners, but did not perform well with DR-RRT. Based on this observation, query relevance appears to be a potentially useful enhancement only for unguided planners.

The filter is also shown to work in nonholonomic problems with reduced efficacy. The most important gains are in terms of the success rates in the `LTunnel` environment, where the filter helps limit the number of unproductive extensions.

4.3 Distance Between Cells

Ideally, the distance between a pair of cells c_1, c_2 would be computed as the minimum inner distance (Section 2.3.3) between any two points $p_1 \in c_1, p_2 \in c_2$, which is an NP-Hard problem. When approximate nearest-neighbors are acceptable, we can employ a rough estimate of the inner distance with a decomposition graph search where the edge weights are the distance between cell centers, measured through the midpoint of the shared facet (Fig. 4.6). While computationally cheap, it unfortunately has many failure cases and is not suitable for exact nearest-neighbors.

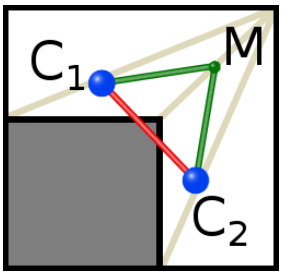


Figure 4.6: Example where the straight-line distance between cell centers c_1, c_2 is not a good metric for the connected workspace distance. A better metric is to measure the path through the shared facet midpoint M. Reprinted with permission from [10], © 2018 IEEE.

When we must have exact nearest-neighbors (i.e. for asymptotically-optimal planning), we require a measure of the inner distance $T_{id}(c_1, c_2)$ between cells. We can accept an approximation $\hat{T}_{id}(c_1, c_2)$ so long as it is upper-bounded by a constant; i.e., $\hat{T}_{id}(c_1, c_2) \leq \delta T_{id}(c_1, c_2)$ for all c_1, c_2 and some $\delta > 0$. With such an approximation, the frontier F will be the set of cells where $\hat{T}_{id}(c_1, c_2) \leq \delta r_L$.

One such approximation is to employ an occupancy grid search by overlaying a grid with voxel length s onto the workspace and mapping cells to voxels with collision checking. An outward manhattan search over the grid from the voxels touching a cell c_1 can then determine the minimum distance to a voxel touching another cell c_2. This yields an approximation with $\delta = \sqrt{2}$ for two dimensional workspaces and $\delta = \sqrt{3}$ for three-dimensional workspaces.

To construct the occupancy grid, we begin by overlaying a grid with voxel length s onto the workspace. The occupied cells are determined by collision-checking each obstacle face against the grid voxels which touch its bounding box. Grid voxels lying in the interior of obstacles will be ignored; we only need the free space and boundary for this purpose.

This grid can be searched with a best-first strategy to estimate the minimum inner distance between two cells c_1, c_2. We first determine the set of voxels which touch each cell. Let V_1 be the set of voxels touching the first cell c_1 and V_2 be the set of voxels touching the second cell c_2. Initialize the distance to each voxel $v \in V_1$ as zero. Mark each $v \in V_1$ as discovered and add each adjacent voxel (through face, edge, or vertex) to the search queue with distance zero.

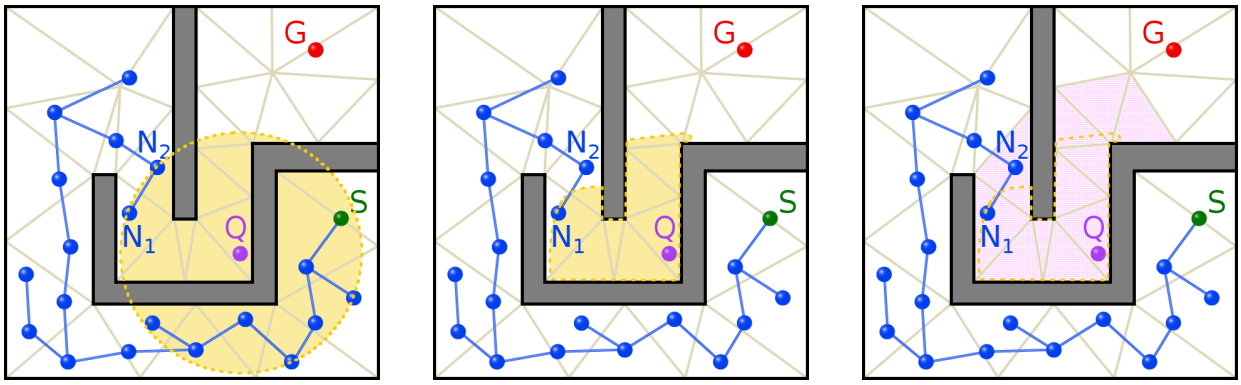
This considers all possible edge cases where a voxel boundary may lie exactly on a cell boundary. Proceed with a best-first search, now considering two voxels as adjacent if they share a face only (i.e. manhattan adjacency). The distance between adjacent voxels will be the voxel length s. This yields an approximation with $\delta = \sqrt{2}$ for two dimensional workspaces and $\delta = \sqrt{3}$ for three-dimensional workspaces (because the greatest error lies along the diagonals). Search outward until discovering the first voxel $v_2 \in V_2$. The distance to v_2 is then the δ-approximate inner distance between c_1, c_2.

4.4 Topological Filtering with Asymptotically-Optimal Planners

Asymptotically-optimal planners such as RRT* and SST require a $\mathcal{C}_{space}$-radius search to identify candidate neighbors. We will refer to the radius required by either planner as r^* in this section. To employ topological filtering with these methods, we must use a radius-based frontier with a carefully chosen radius to capture the necessary portions of $\mathcal{C}_{space}$.

For these radius-based nearest-neighbor methods, we will select a topological frontier beginning at the initial cell c containing the query point q and include all cells which have a minimum distance to c of less than or equal to a radius value. The filter's radius must be carefully selected to respect the differences between the $\mathcal{C}_{space}$ and workspace distances, as we describe in this section.

Preserving the optimality guarantees for either planner requires that the nearest-neighbor operation locate all configurations within r^* of a query configuration $q \in \mathcal{C}_{space}$. For a distance metric $D(q_1, q_2)$ which measures the proximity of two configurations q_1, q_2 through $\mathcal{C}_{space}$, this *radius criterion* indicates that all roadmap configurations $x \in \mathcal{C}_{free} \mid D(q, x) \leq r^*$ must be considered as candidates. However, this choice of radius is motivated by percolation arguments from random geometric graph theory which do not account for obstacles [22]. In the presence of obstacles, we observe that the graph must percolate over $\mathcal{C}_{free}$ rather than $\mathcal{C}_{space}$. We will thus bend the distance requirement to consider a radius measured with respect to the inner distance in $\mathcal{C}_{free}$, denoted as $D_{id}(q_1, q_2)$. The inner distance measures the length of the shortest path from q_1 to q_2 without leaving $\mathcal{C}_{free}$ (Figs. 4.7(a),4.7(b)). It stands to reason that this more accurately addresses the percolation requirement that all vertices within a particular radius are connected because connectivity



(a) (b) (c)

Figure 4.7: Relationship between $\mathcal{C}_{space}$ radius, inner distance, and topological frontier. (a) A $\mathcal{C}_{space}$ radius (yellow) as computed by a standard distance metric, projected onto workspace. (b) The same radius measured by inner distance. (c) The radius-based frontier (purple) selected to model the desired inner distance.

must occur with respect to this metric.

However, we cannot feasibly compute the set of configurations within a $\mathcal{C}_{space}$ ball β_{r^*} with respect to D_{id}. Our strategy then will be to leverage the workspace topology to approximate a projection of β_{r^*} into the workspace W, where we can then leverage our topological map to locate a superset of the desired candidates. Formally, let β_{r^*} be a ball of radius r^* in the metric space $M_{\mathcal{C}} = \{\mathcal{C}_{space}, D_{id}\}$ which contains some set of configurations $X_{\mathcal{C}} \subset \beta_{r^*}$. Let β_{r_L} be a ball in the metric space $M_W = \{W, T_{id}\}$, where T_{id} represents the inner distance in W and β_{r_L} contains a set of configurations X_W. We aim to compute an approximate mapping $\phi : M_C \rightarrow M_W$ such that the co-image of a ball in M_W is a superset of its counterpart in M_C, e.g. $\phi^{-1}(\beta_{r_L}) \supset \beta_{\mathcal{C}}$. When this is possible, the topological filter is able to provide a reasonable over-estimate of the configurations in β_{r_L} which satisfy the radius criterion as measured through $\mathcal{C}_{free}$ by D_{id}.

To satisfy this requirement, we can leverage a relationship between the $\mathcal{C}_{space}$ metric $D_{id}(q_1, q_2)$ and the workspace metric $T_{id}(q_1, q_2)$. If $T_{id}(q_1, q_2) \leq \alpha D_{id}(q_1, q_2)$ for all $q_1, q_2 \in \mathcal{C}_{free}$ and some constant $\alpha > 0$, then all configurations within a D_{id}-distance of r^* are also within an T_{id}-distance of $r_L = \alpha r^*$. In this case, we can determine the candidate neighbors which satisfy the radius criterion using a distance measurement in workspace. This relation between D_{id} and T_{id} holds for

many popular distance metrics, including the canonical L^2 norm. For mobile-base robots, it can be satisfied with $\alpha = 1$ since the L^2 norm will always be at least the translational displacement. This is an approximation of the ideal mapping between spaces ϕ that is sufficient to guarantee the desired co-image property.

For satisfying distance metrics, we can employ a topological frontier F which includes all cells within inner distance of αr^* from the cell c containing a query point q, where inner distance between cells c_1, c_2 indicates the minimum possible inner distance between any pair of points in each cell (Fig. 4.7(c)). This frontier describes a radius in connected workspace. We argue that this set F will contain all configurations X_W, and therefore all configurations $X_\mathcal{C} \subset X_W$ because the relationship between metric spaces establishes a valid approximation of the ideal mapping ϕ.

In simpler terms, the set F includes all configurations which may be connectable to q with a path through $\mathcal{C}_{free}$ of length r^* or less. The radius criterion in optimal planners is specifically meant to lower-bound this path distance, so our slight over-estimate will yield asymptotically-optimal behavior. Any configurations that are within an absolute D-distance of r^* to q with D_{id}-distance greater than r^* are necessarily occluded by an obstacle and will not be connectable to q.

4.4.1 Validation

To validate the filter with asymptotically-optimal planners, we demonstrate its performance in two environments with RRT* and SST. We compare the filtered version against the same algorithms using brute-force nearest-neighbors (nearest) and k-d tree for the non-radial search in RRT*.

4.4.1.1 Experiment Setup

The environments are two examples from the feasibility-planning experiments (`Helico` and `LTunnel`) that were solvable by the unguided methods (Section 4.2.2) to ensure that the difference between the guided and unguided algorithms during path refinement (i.e., after the initial plan is found) can be compared. As in the feasibility tests, `Helico` represents a more open environment while `LTunnel` represents a problem with three moderately narrow passages.

The methods all employ a $\mathcal{C}_{space}$ euclidean distance metric with straight-line local planning and

uniform random sampling in $\mathcal{C}_{free}$. Each variation is run thirty times to 20k iterations. We report the generated map size, running time, nearest-neighbor time, path cost, and success rate (where success is defined as generating a path before the iteration limit). The plots show an average as well as the minimum/maximum envelope for each method as a function of iteration count. Times are reported in seconds and path cost in euclidean distance. Only successful trials are plotted.

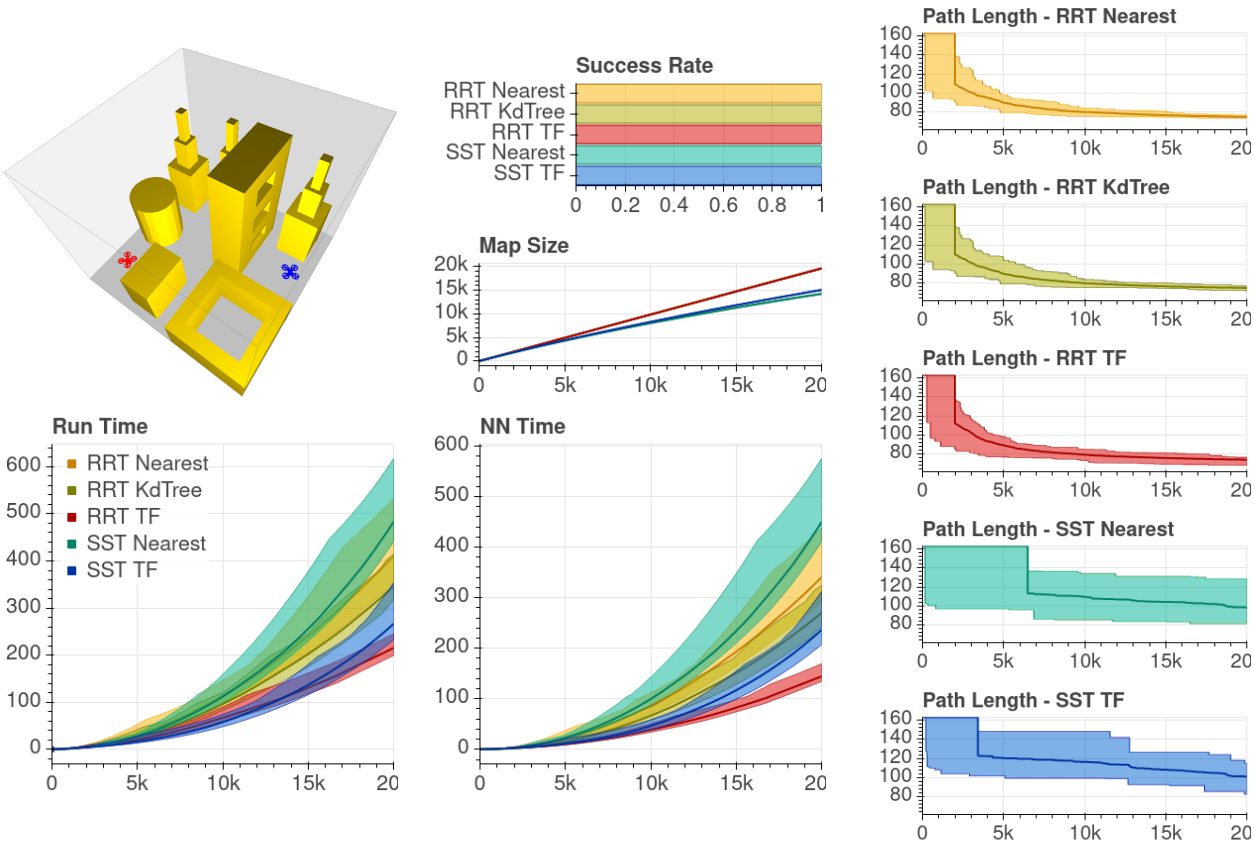


Figure 4.8: Evaluation of the topological filter in a holonomic free-body problem where a quadcopter must traverse a cityscape. The x-axis shows iterations, while the y-axis shows vertex count, time in seconds, or path cost measured by L^2 norm. The solid lines show the average while the colored areas represent the spread.

All experiments were executed on a desktop computer running CentOS 7 with an Intel® Core™ i7-3770 CPU at 3.4 GHz, 16 GB of RAM, and the GNU g++ compiler version 9.2.0. The workspace tetrahedralization was performed with a combination of the TetGen [37] and CGAL [52]

libraries.

4.4.1.2 Analysis

In the `Helico` problem (Fig. 4.8), we see that the filter provides consistently faster run and nearest-neighbor time for both methods vs. the brute-force and k-d tree versions. The gain is sufficient that the envelope maximum for the filtered algorithms performs better than the minimum for the non-filtered instances. We also observe no loss in path quality, and that the filtered algorithms take equal or less time for all trials to discover an initial path.

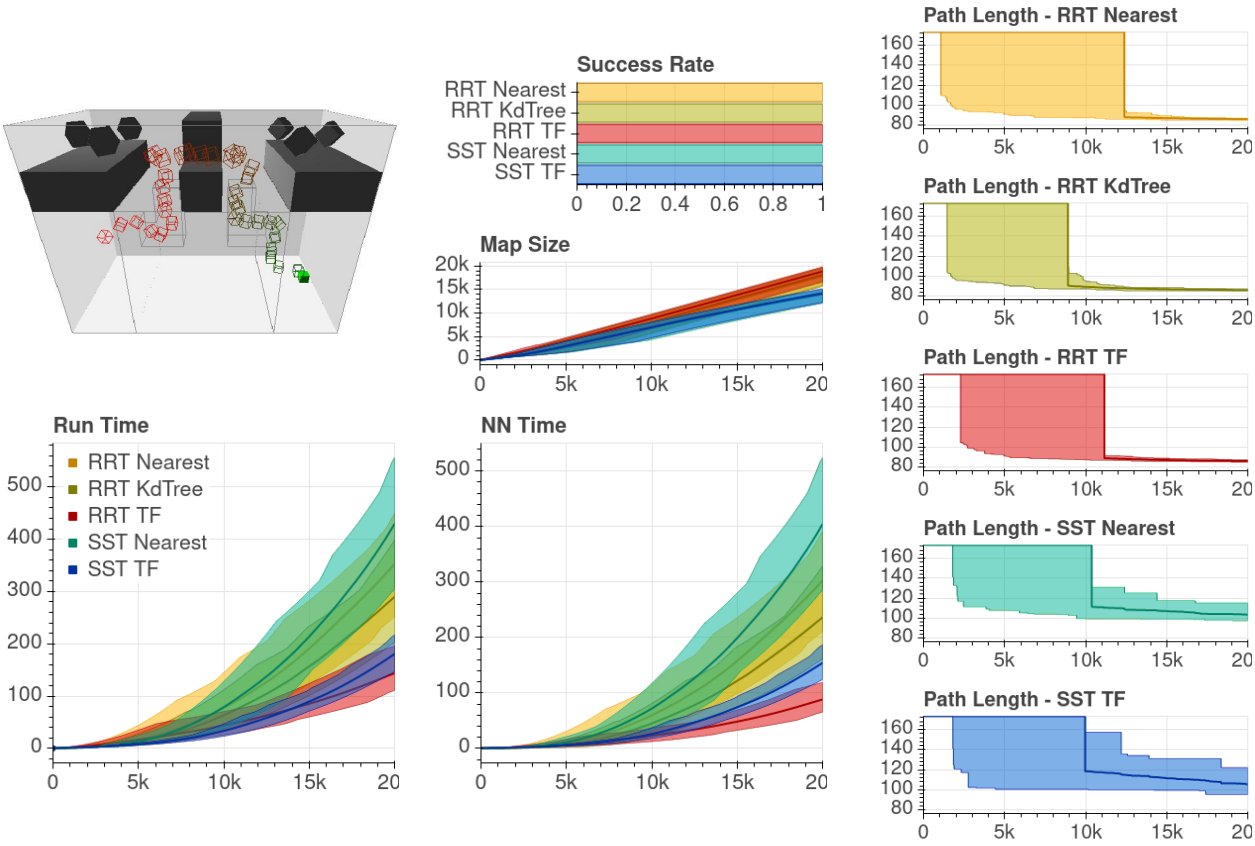
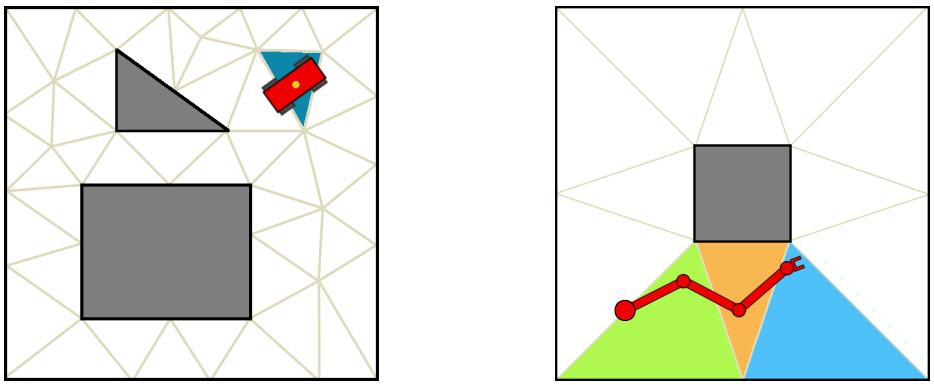


Figure 4.9: Evaluation of the topological filter in a holonomic free-body problem where a box robot must traverse two L-shaped tunnels and a narrow gap. The x-axis shows iterations, while the y-axis shows vertex count, time in seconds, or path cost measured by L^2 norm. The solid lines show the average while the colored areas represent the spread.

In the `LTunnel` problem (Fig. 4.9), we observe a similar trend: the filtered algorithms' maximum envelope lies beneath the unfiltered variants' for both time metrics. There is no loss in average path quality for either RRT or SST, although the filtered SST exhibits slightly higher variance than the unfiltered version. The number of iterations until all paths solve is also slightly higher for the filtered RRT in comparison to the k-d tree version, but the filter's advantage in run time means that it will reach that point before the k-d tree would.

In both problems, we see a clear advantage in both run-time and nearest-neighbor time for the filtered methods which continues to increase as the iteration count grows. This supports the hypothesis that the filter's efficiency and efficacy grow with roadmap size, which is a very desirable property in asymptotically-optimal planning.



(a) Single-link Containment (b) Multi-link Containment

Figure 4.10: Configuration containment in decomposition cells. (a) A configuration of a single-link robot (red) is contained by the decomposition cell (blue) which contains its reference point (yellow). Reprinted with permission from [10], © 2018 IEEE. (b) A configuration of a multi-link robot (red) where the neighborhood key cells are colored for the first (green), second (orange), and third (blue) joints.

4.5 Extension to Manipulators

Applying the topological filter to multi-link robots presents an obvious question on whether a single reference point p_R is adequate to characterize the workspace neighborhood of the full

robot body. A single p_R on the robot's base will certainly provide a valid filter for mobile-base manipulators, but this fails to capture connectivity for the remainder of the links. The same issue applies to fixed-base manipulators where the end-effector is the most important body. In these cases, there is no choice of p_R that provides a complete topological mapping for the entire robot.

The topological filter can be extended to support filtering on multiple links of the robot by generalizing the concept of a cell containing a configuration. Let $B = \{b_1, b_2, \ldots, b_n\}$ be an ordered tuple of the robot R's n individual component links. When R is positioned at some configuration q, each of its individual links $b_i \in B$ will be contained by some cell $c \in W$, where W is the set of all cells in the decomposition graph. We define the *neighborhood key* of R at a configuration q as the ordered tuple of cells $\{d_1, d_2, \ldots, d_n\}$ occupied by R's individual links B when R is positioned at q (Fig. 4.10(b)).

The set of all valid neighborhood keys and transitions between them is exceedingly large even for a small decomposition. Each link $b_i \in B$ may be contained by any cell $d_i \in W$, and multiple links may also be contained by the same cell such that $d_i = d_j$ for $i \neq j$. The number of possible neighborhood keys is thus $|W|^n$, so we unfortunately cannot form a graph over that space. However we can approximate the function of such a structure by composing a separate topological map for each of the robot's links.

To apply topological filtering to multiple links of R, we can construct a separate topological map for individual links. When searching for a nearest-neighbor to some configuration q, the filter now begins by locating the neighborhood key for q via the set of individual maps. The frontiers and candidates are then identified as in the rigid-body case and joined by a soft intersection to produce a refined candidate set.

To compute the soft intersection, we count the number of times each candidate appears across all frontiers and select the candidates which have been observed most frequently. This identifies the best available candidates and empirically produces good results. It is achievable with the same time complexity as a strict intersection, which is undesirable because it may produce an empty candidate set. This can occur due to a relatively high mobility of the end-effector (and other distal

links) in comparison with the base (and other proximal links). As the number of links and their lengths increase, this problem worsens because there are a greater variety of configurations which do not simultaneously occupy the topological frontiers for all links. An empty intersection a worst-case scenario for the filter because all of the computation which was performed to locate candidates is wasted without discovering any useful information.

We also note that it is not strictly necessary to filter on all of the links, as there is a kinematic relationship between them defined by the properties of their adjacent joints. Depending on the mobility of the robot, it may be desirable to filter only one link or some subset of the links, such as the base, end-effector, and a middle elbow. The filter will ignore the unfiltered links in this case and only restrict the candidate set based on the links under consideration.

4.5.1 Asymptotically-Optimal Filtering with Multi-link Robots

For multi-link robots, we require a radius-based frontier for each joint. For fixed-base robots this is feasible by establishing a maximum ratio of workspace translation to $\mathcal{C}_{space}$ distance to define an α value for each link to be filtered. For mobile bases however, this presents a challenge as there is often no meaningful way to bound the ratio for links other than the base. In such cases, a filter on the base body alone will preserve the AO properties but lack the discerning power of a filter covering more links. The filter power can be increased by applying a filter with a carefully selected radius to other links, but this is a heuristic which unfortunately can't strictly preserve the AO properties of RRT* (although the near-AO properties of SST will be preserved). Because this may rule out configurations that are truly within the optimal radius r^*, the resulting planner may not converge to the true optimal path and is thus only near-optimal.

Despite the theoretical limitations, we observe good results in practice from applying the filtering radius for the base to the end-effector alone. This prefers to attempt rewiring between more likely candidate configurations at the cost of true asymptotic optimality.

In some cases however, a workspace-based filter on the end-effector is counter productive. This occurs for end-effector positions where the robot has high redundancy and thus self-mobility. High self-mobility means that there are many possible configurations which can place the end-

effector at or near the same position, and it is not necessarily the case that transitioning between all of these configurations is easy or likely. For very highly redundant manipulators, it is best to perform filtering on links that are more proximal to the base with lower self-mobility, or to at least include such a link in a composite multi-link filter to avoid this problem. This sacrifices some discriminating power but avoids over-fitting the neighbor selection on the assumption that configurations with nearby end-effectors are easy to transition between.

4.5.2 Validation

To validate the method, we compare RRT* and SST with brute-force, k-d tree, and topological filtering over three manipulator planning problems with thin walls that extend beyond feasibility planning and into path refinement. The k-d tree variant employs a k-d tree only during the 1-nearest check and not the radial searches, and is thus not applicable to SST.

4.5.2.1 Experiment Setup

The problems include a fixed-base manipulator maneuvering between shelves (Fig. 4.11), a mobile-base manipulator moving around shelves (Fig. 4.12), and a mobile-base Kuka Youbot loading a lumber beam from a shelf into a shipping container (Fig. 4.12). This covers a range of manipulator problems where the topological filter is expected to provide some level of utility by avoiding candidate neighbors likely to intersect with thin walls. In all cases, the filtering is performed only on the end-effector to maximize this potential benefit (which also implies that the RRT* variants are limited to asymptotic near-optimality).

The same robot is used for the fixed-base and mobile-base problems. It has four spherical joints and is permitted to translate but not rotate its base in the mobile version (because rotation would be redundant with the first joint). This makes for a total of eight DOF for the fixed problem and ten for the mobile one. The Kuka Youbot has five revolute joints and is permitted to both translate and rotate its base, for a total of eight DOF.

The methods all employ a $\mathcal{C}_{space}$ euclidean distance metric with straight-line local planning and uniform random sampling in $\mathcal{C}_{free}$. Each variation is run thirty times to 20k iterations. We report

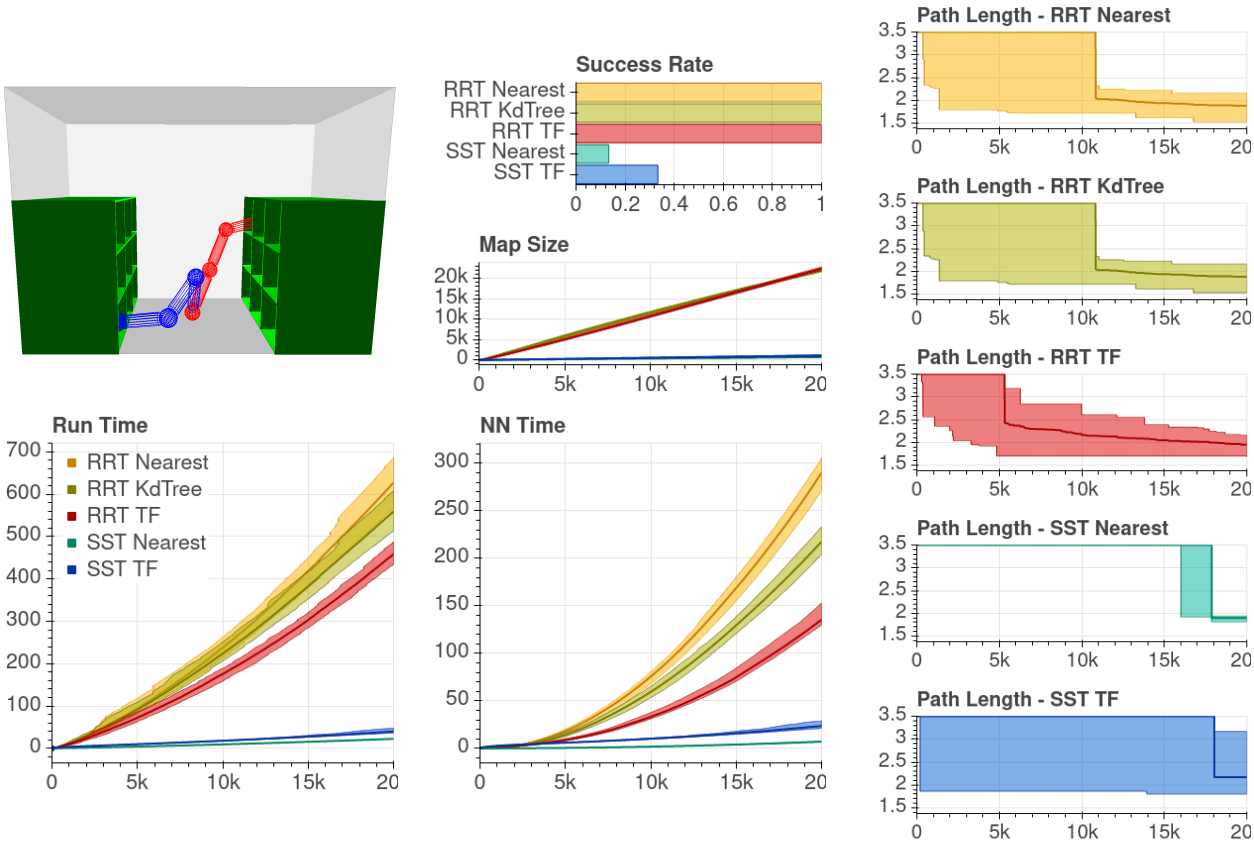


Figure 4.11: Evaluation of topological filtering in a fixed-base manipulator problem where the robot must transition between grasping positions within pick shelves on alternate sides of the workspace. Thirty trials were performed, and time plots include only successful runs which created a path by 20k iterations. The x-axis shows iterations, while the y-axis shows vertex count, time in seconds, or path cost measured by L^2 norm. The solid lines show the average while the colored areas represent the spread.

the generated map size, running time, nearest-neighbor time, path cost, and success rate (where success is defined as generating a path before the iteration limit). The plots show an average as well as the minimum/maximum envelope for each method as a function of iteration count. Times are reported in seconds and path cost in euclidean distance. Only successful trials are included in the map size, time, and cost plots.

All experiments were executed on a desktop computer running CentOS 7 with an Intel® Core™ i7-3770 CPU at 3.4 GHz, 16 GB of RAM, and the GNU g++ compiler version 9.2.0. The workspace tetrahedralization was performed with a combination of the TetGen [37] and CGAL [52]

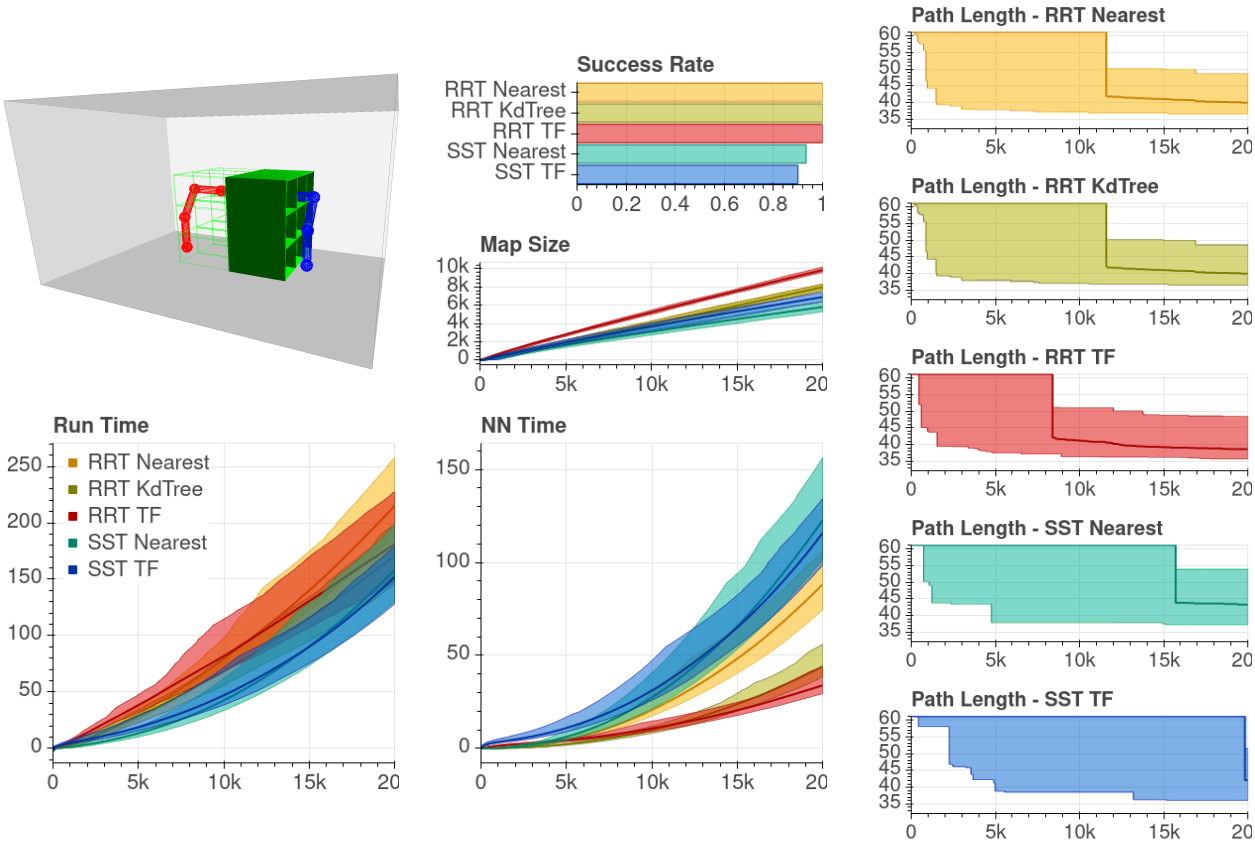


Figure 4.12: Evaluation of the topological filter in a mobile-base manipulator problem where the robot must transition between grasping positions within back-to-back pick shelves by translating around the exterior. The reverse shelf is shown in wire-frame for visual clarity of the goal position. Thirty trials were performed, and plots include only successful runs which created a path by 20k iterations. The x-axis shows iterations, while the y-axis shows vertex count, time in seconds, or path cost measured by L^2 norm. The solid lines show the average while the colored areas represent the spread.

libraries.

4.5.2.2 Analysis

In the fixed-base problem (Fig. 4.11), we observe reasonable benefits from the filter in all metrics of interest. The execution and nearest-neighbor time envelopes for the filtered RRT* are clearly below the unfiltered versions, and the path cost converges more quickly and to a lower minimum value. The SST variants on the other hand struggle with this problem because the notion of a witness radius does not work well with fixed-base manipulators. The problem here is that

the robot's $\mathcal{C}_{space}$ is entirely comprised of its joint space, and small changes in joint space values can yield drastically different changes in the end-effector position. This means that small motions of the robot may be rejected due to a lower-cost witness 'nearby' even though the configurations are significantly different. The witness regions are meant to define small neighborhoods in $\mathcal{C}_{free}$ where the contained configurations have similar connectivity properties, but this breaks down for joint space due to the large differences in semantics that can occur with small differences in the $\mathcal{C}_{space}$ metric.

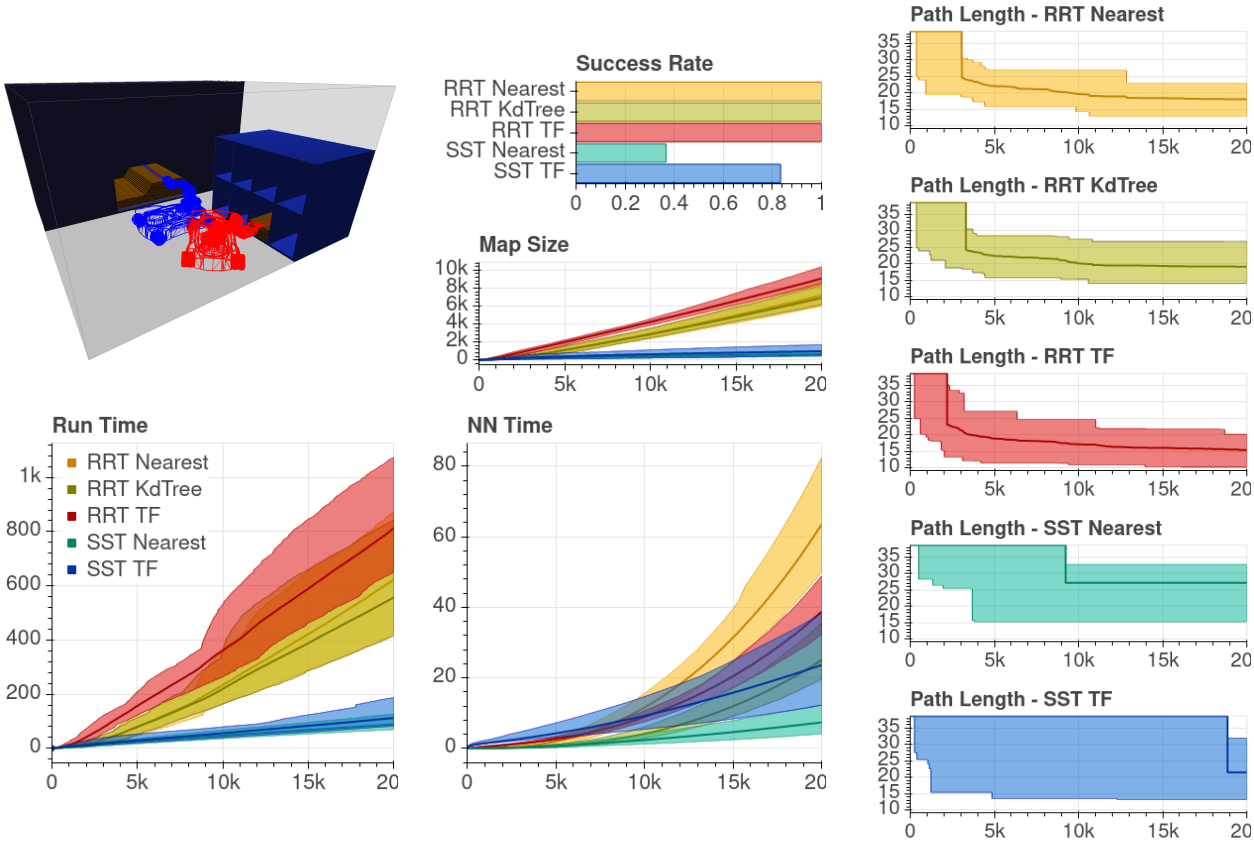


Figure 4.13: Evaluation of the topological filter in a mobile-base manipulator problem where a Kuka Youbot must move a lumber beam from a shelf into a truck bed. Thirty trials were performed, and plots include only successful runs which generated a path by 20k iterations. The x-axis shows iterations, while the y-axis shows vertex count, time in seconds, or path cost measured by L^2 norm. The solid lines show the average while the colored areas represent the spread.

In the mobile-base problem (Fig. 4.12), we observe that the SST problems are mitigated by the influence of the translational DOFs on the distance metric. However, the filtered SST algorithm under-performs compared to the unfiltered version in convergence rate; it does achieve equivalent path cost but takes longer to do so, with more trials failing to discover a path by the iteration limit. This occurs despite a better extension success rate indicated by the larger map size for the filtered version. For RRT*, we also observe a higher rate of successful extension and comparable path cost, but with quicker discovery of initial solution. The filtered algorithm's nearest-neighbor time is roughly comparable to the k-d tree version despite working with a 25% larger roadmap.

The Kuka problem (Fig. 4.13) again shows the RRT* variants generating larger maps, this time with a significant performance loss in nearest-neighbor time vs. the k-d tree version. In this problem, the filter is only useful during the initial motion to remove the beam from the storage bins. It clearly helps SST during this part of the problem, where the unfiltered version struggles to overcome even a modest witness radius with the small motions required to ease out of the initial constrained position. We also observe that both filtered algorithms produce better path cost than their unfiltered counterparts, although the difference is not significant given the envelope spread.

These problems demonstrate that the filtering concept can be applied to asymptotically-optimal manipulator problems. Generally we observe that the step down to asymptotic near-optimality with RRT* by filtering on the end-effector for mobile base problems provides a better convergence rate to lower path cost despite the reduced guarantee. This occurs because filtering during the extension step changes the shape of the tree to avoid excessively driving configurations up against obstacle walls, causing more of the extensions to be again extendable in a subsequent iteration. This also causes the higher rate of extension success observed by the larger maps generated by the filter. Filtering during the rewiring step helps manage the cost of rewiring the larger map, and avoids costly local plans which are unlikely to succeed.

4.6 Isolated Evaluation

The validation experiments presented thus far have been within the context of a planning problem, which is the intended use case for the topological filter. While this demonstrates the methods

efficacy as a planning tool, it is less clear in differentiating its behavior from other nearest-neighbor methods because the selection of different nearest-neighbors leads to changes in the roadmap's growth; over many iterations, the roadmaps built with and without filtering diverge and become quite distinct. It is thus difficult to determine whether the filter is producing computational advantages or purely benefits the evolution of the roadmap. Here we compare the topological filter to other nearest-neighbor methods in a setting that removes this source of ambiguity.

An ideal nearest-neighbor method (either k or radius) would return the closest set of configurations that are connectable by the local planner to a query point q. We will denote such an ideal set as the *oracle neighbors* for q. Our goal here is to measure how well a given nearest-neighbor method identifies the oracle neighbors and excludes unconnectable options.

We define two metrics that measure this behavior. The *oracle selection* is the fraction of oracle neighbors returned; this indicates the method's ability to find the ideal choices, with a high value indicating strong performance. The *unconnectable rejection* is the fraction of neighbors returned that are connectable by the local planner. A high value for this metric indicates that the method is effective at filtering out choices which do not lead to a successful local plan.

An ideal nearest-neighbor method would produce a value of one for both metrics, indicating that it both found all of the ideal choices and returned no choices that were infeasible. A high value for selection with a lower rejection score indicates an algorithm that did find the ideal choices, but also located undesirable candidates. A low selection and high rejection is also possible if the algorithm finds many connectable neighbors, but not the nearest ones included in the oracle set.

It is important to note that these metrics are specific to the given local planner, as different choices of local planner will yield different paths through $\mathcal{C}_{space}$ and thus different connectivity. They can thus be viewed as ways of measuring the nearest-neighbor method's fitness for that particular local planner.

4.6.1 Experiment Setup

We begin by generating a roadmap with a fixed number of samples and determining the nearest connectable neighbors (either k or radius) to each configuration by brute force. After determining

the oracle set, we run several nearest-neighbor methods for each configuration and evaluate both oracle selection and unconnectable rejection. We then add additional samples to the roadmap and repeat the test to show how the outcome changes with increasing roadmap density.

We perform this experiment on a set of environments with varying levels of occlusion to characterize how the filter behaves in a range of $\mathcal{C}_{space}$ topologies. The `Open` problem presents a completely open space, while `Maze` exhibits narrow tunnels and thin walls (Fig. 4.3(c)). We show two rigid-body robots, one a three DOF sphere and the other a six DOF stick which cannot turn around in the maze. The sphere's radius is equal to the cross-sectional radius of the stick.

The sphere characterizes the filter's behavior in a case where it can map the full $\mathcal{C}_{space}$. This robot is essentially an inflated point, and the mapping is complete because the $\mathcal{C}_{space}$ for a point robot is exactly the workspace. The sphere's $\mathcal{C}_{space}$ is thus an 'inflated' version of the workspace produced by the Minkowski sum of the robot body with each obstacle. The stick shows how this changes when only a partial mapping is possible. In this case, the added rotational dimensions preclude a direct relationship between the two spaces. The indirect relationship is easier to conceive by considering a problem for a robot that translates (without rotation) in three dimensions x, y, z and, a topological filter that considers only the x, y plane. Here the filter can rule out configurations that are far away in the x, y dimensions but has no information about the z component. This allows for rejection of poor candidates relative to x, y but not z. Analogously, the workspace filter for the stick robot allows for good rejection relative to x, y, z but not the rotational components.

We show evaluations with both k-nearest and radius methods. For the topological filter we show two evaluations: the 'Frontier' version examines all topological candidates, while the 'TopologicalNF' version examines the candidates selected by the underlying nearest-neighbor method. The former evaluates the two connectivity metrics on the identified topological candidates, while the latter evaluates the final selection that will be used in practice. Since the Frontier includes a subset of the TopologicalNF output, in all cases we should observe that the Frontier's selection is at least as great as TopologicalNF's. Rejection may vary as the underlying method can still make poor choices.

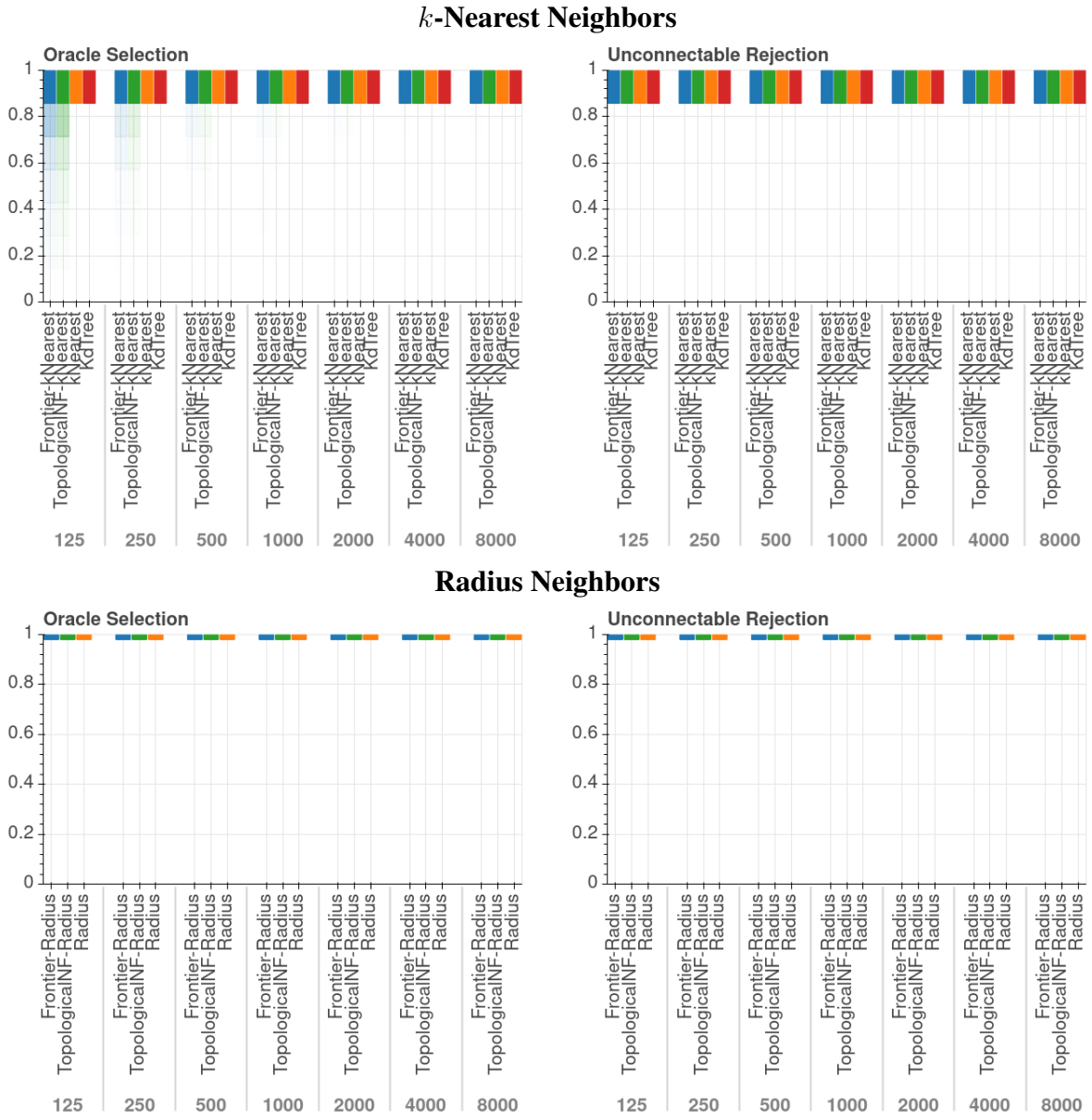


Figure 4.14: Isolated evaluation of the topological filter in the `Open` problem with a sphere robot.

Thirty-five trials were performed for each experiment. In all cases, we employ a euclidean distance metric and straight-line local planner. The k-nearest variety uses eight nearest neighbors. We plot the selection and rejection for each method and sample size with a color-hue probability distribution, where darker colors indicate higher probability. The k-nearest plots are discretized into eight segments to enhance visibility, as the values will always be fractions of eight. The radius plots are presented with fifty segments to reflect the much larger number of candidates under

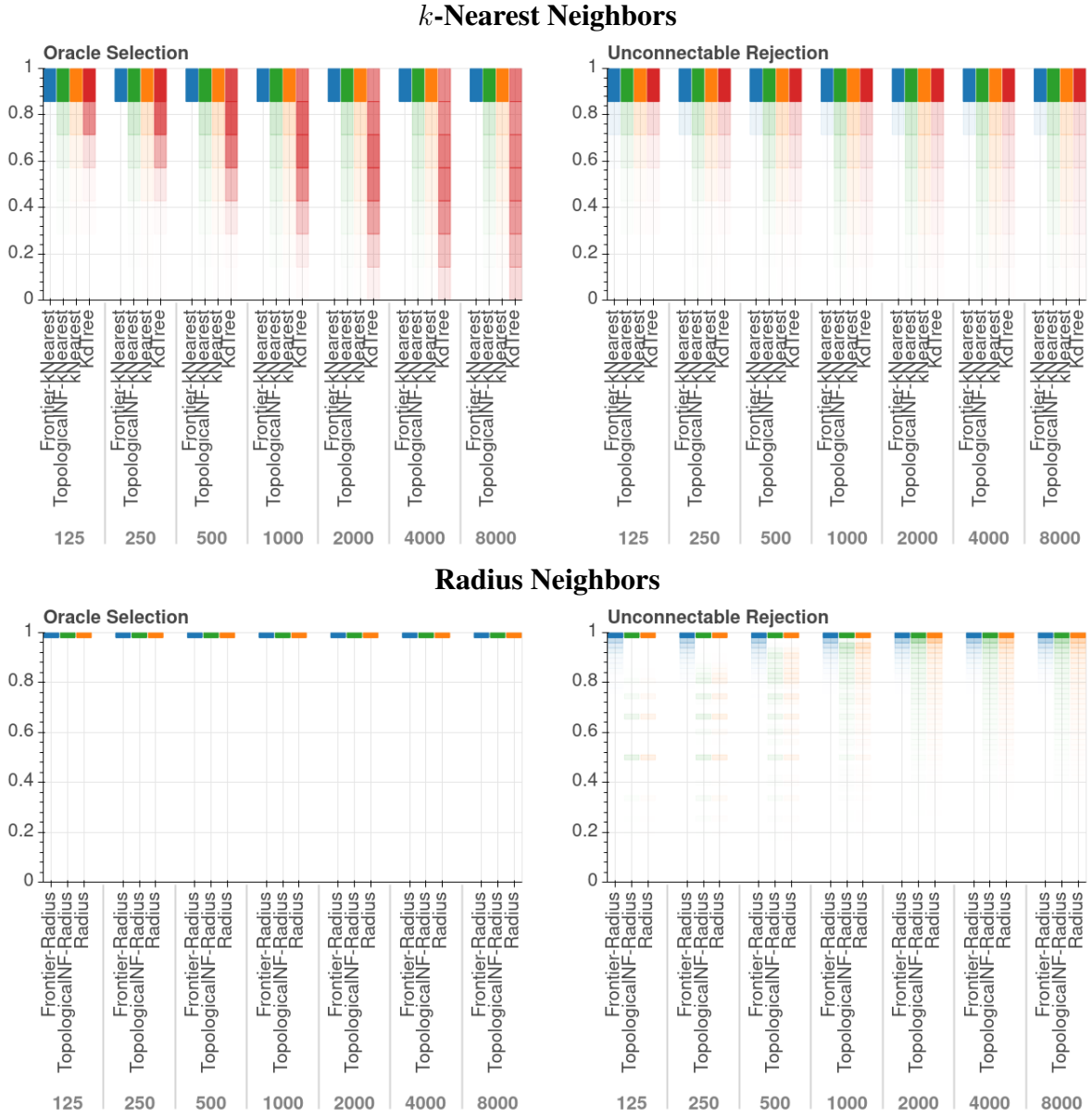


Figure 4.15: Isolated evaluation of the topological filter in the `Open` problem with a stick robot.

consideration.

All experiments were executed on a desktop computer running CentOS 7 with an Intel® Core™ i7-3770 CPU at 3.4 GHz, 16 GB of RAM, and the GNU g++ compiler version 9.2.0. The workspace tetrahedralization was performed with a combination of the TetGen [37] and CGAL [52] libraries.

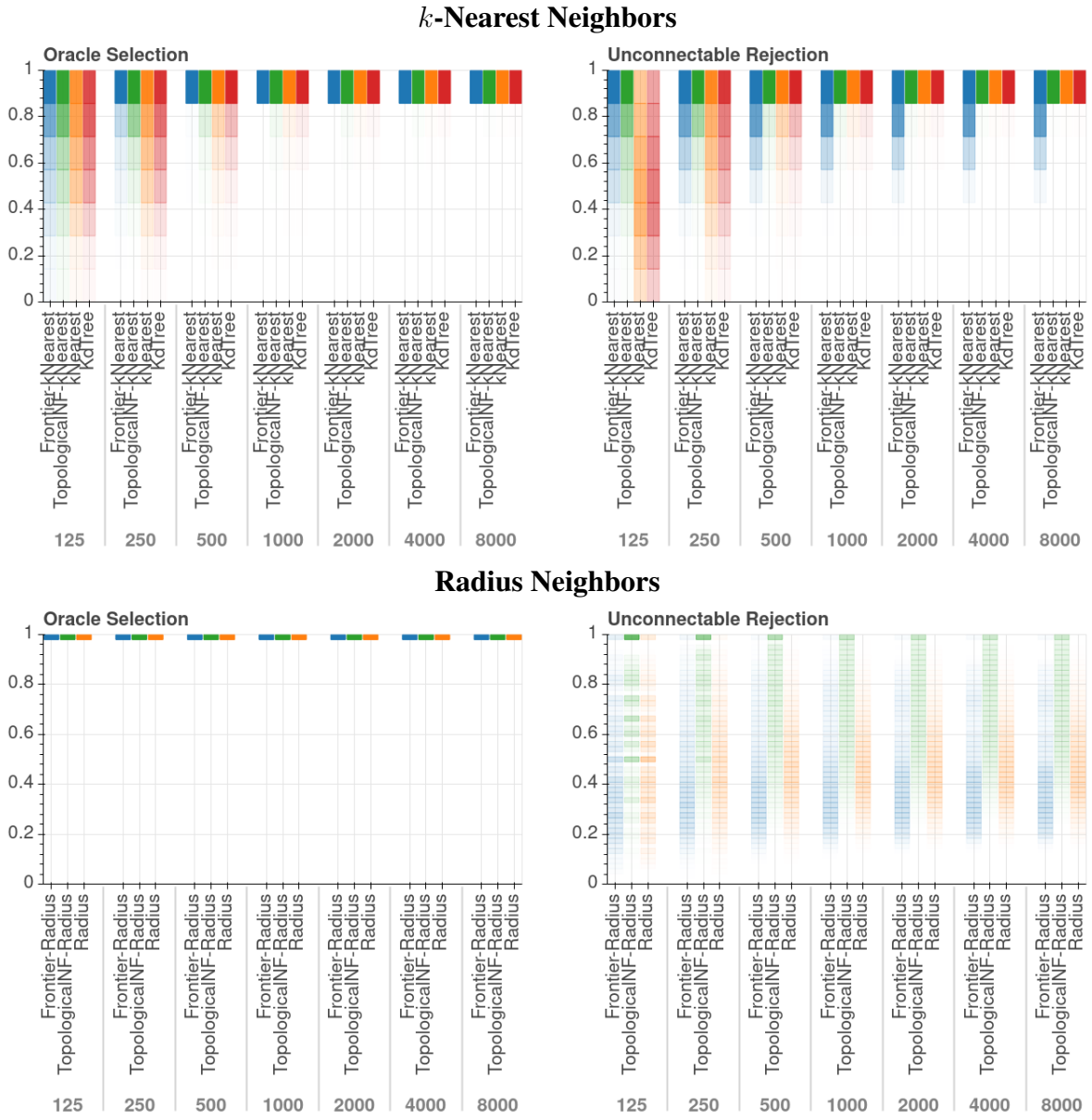


Figure 4.16: Isolated evaluation of the topological filter in the `Maze` problem with a sphere robot.

4.6.2 Analysis

In the `Open` problem, the sphere robot (Fig. 4.14) exhibits perfect selection and rejection in all cases except for the filter at low sample counts, which is expected given the lack of obstacles and the convex, non-rotational nature of the robot. Imperfect selection occurs for the filter at low sample counts because the samples are more dispersed through the environment, and the nearest

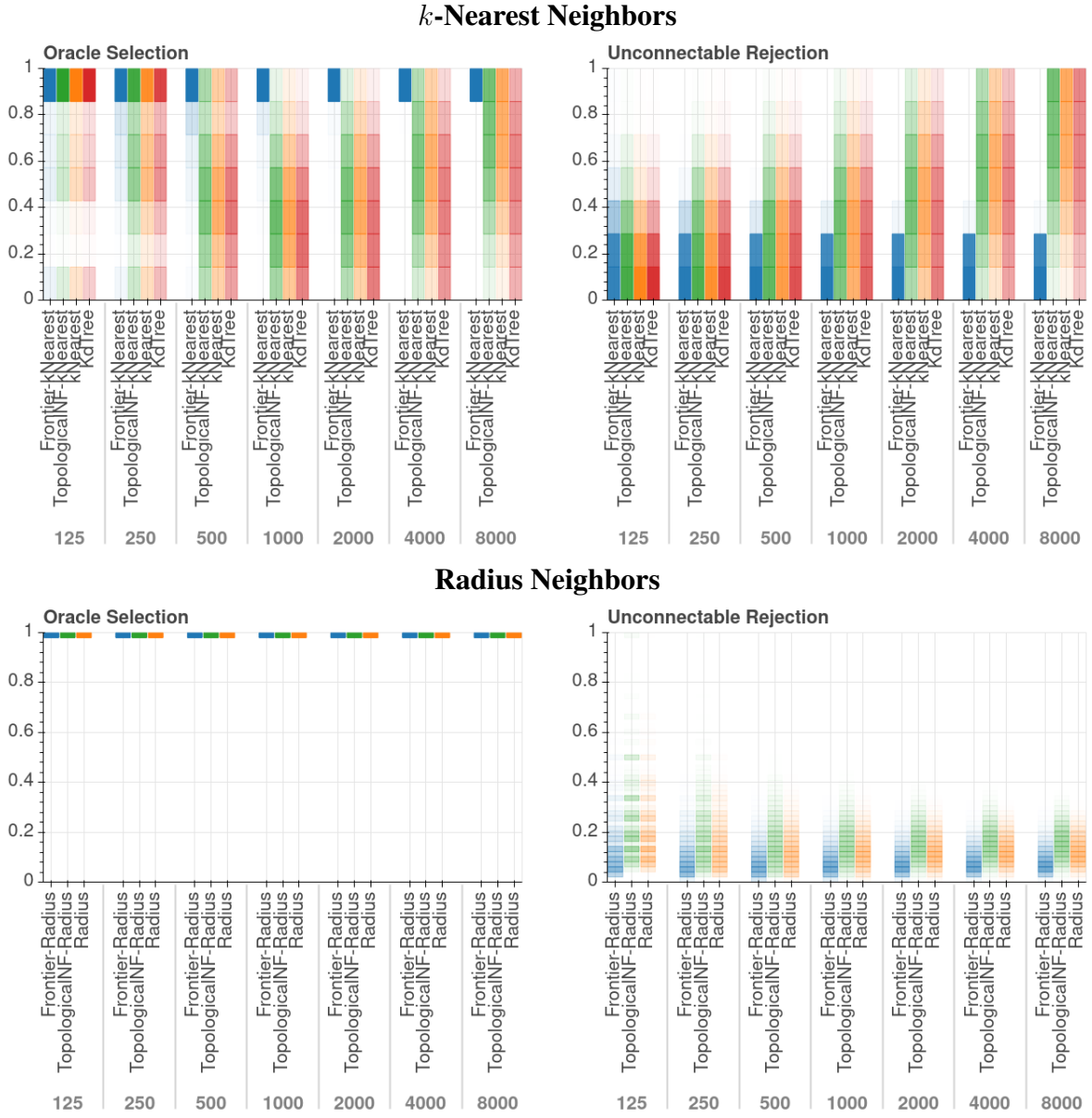


Figure 4.17: Isolated evaluation of the topological filter in the `Maze` problem with a stick robot.

neighbors to the first discovered vertex lie farther away than the backtrack distance. This problem vanishes by 500 samples as the roadmap is sufficiently dense that there are always at least eight neighbors within a backtrack distance of the first discovered candidate.

The stick robot shows a slightly different picture (Fig. 4.15). Radius selection is ideal, but rejection is not always so. This occurs near the environment boundary, where some configurations would cross the border when attempting a straight-line plan through $\mathcal{C}_{space}$. In the k-nearest ver-

sion, we see that the frontier reliably selects the oracles, but the subsequent selection of the eight nearest in the TopologicalNF method can still miss parts of the oracle set. This again occurs near the environment boundary, where a local plan from the very nearest candidates may collide with the border. We also observe that the k-d tree rather frequently fails to select oracles despite good rejection, which is due to the approximate nature of the method and the higher dimension of the stick robot's $\mathcal{C}_{space}$ compared with that of the sphere.

In the `Maze` problem, the sphere robot exhibits lower selection rates overall in the k-nearest version due to the presence of obstacles (Fig. 4.16). While all methods approach ideal selection at high sample counts, we observe that the filter converges more rapidly than the others due to its awareness of connectivity. The filter's rejection also increases more rapidly for the same reason. The radius version shows ideal oracle selection, which is again expected because each method identifies at least all candidates within the radius region. The filter additionally shows strong rejection performance in this case due to its ability to ignore configurations across the walls.

While the sphere robot illustrates a relatively ideal mapping between the filter frontier and the robot's $\mathcal{C}_{space}$, the stick robot shows how performance changes when this mapping is less complete (Fig. 4.17). In the k-nearest problem, we observe lower performance in both metrics and a more even spread across the methods. The filter has a minor advantage in selection at lower sample counts, but this disappears as the roadmap gains density and the eight-nearest neighbors are likely to be within the same workspace neighborhood anyway. The filter still provides computational advantages in that case by locating the neighbors without looking at the entire roadmap. In the radius version, we see that the filter has a marginally better rejection rate, roughly 30% better than the standard radius method. This amounts to a significant savings in asymptotically-optimal settings, especially for configurations that are very close to the boundary.

4.7 Summary

We describe the topological filtering algorithm for nearest-neighbor search, and show experiments which demonstrate that it both improves the likelihood of successful extension and reduces the computational cost of the nearest-neighbor process. These benefits arise from the use

of a workspace model to approximate the connectivity of $\mathcal{C}_{free}$, thereby providing some level of obstacle-awareness to the planner during nearest-neighbor selection.

5. SUMMARY AND CONCLUSIONS

We show two methods which leverage workspace topology in sampling-based planning. Dynamic Region sampling guides the generation of new configurations along a topological skeleton known to encode partial solutions, and topological filtering sifts nearest-neighbor candidates to exclude unlikey connections to configurations which are close in $\mathcal{C}_{space}$ but far away through the shortest path in $\mathcal{C}_{free}$. The driving insight behind both methods is to exploit information about how regions of workspace are connected to estimate the topology of $\mathcal{C}_{free}$, and to focus a sampling-based planner's resources on the areas of the domain which are likely to be productive. These methods provide a higher-level view of the problem which enables a planner to make more strategic choices about the best way to explore $\mathcal{C}_{free}$.

An avenue for furthering the work is to investigate whether either method can be applied with subspace models for something other than the translational DOFs, for which we have used the workspace as a natural representation. Theoretically, the techniques presented here are not restricted to functioning with the translational subspace: any meaningful subspace should suffice. It is also likely that derived spaces will work, such as the end-effector configuration for a manipulator, which is a manifold derived from the full configuration space of the robot. The critical element in both cases is to derive meaningful boundaries in the space where guidance will occur. This enables construction of a topological skeleton for dynamic region sampling or a cell decomposition for the topological filter.

REFERENCES

[1] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Proc. IEEE Symp. Found. Comp. Sci. (FOCS)*, (San Juan, Puerto Rico), pp. 421–427, Oct. 1979.

[2] J. F. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.

[3] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Trans. Comput.*, vol. 100, no. 2, pp. 108–120, 1983.

[4] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin, "On finding narrow passages with probabilistic roadmap planners," in *Proc. Int. Wksp. Alg. Found. Robot. (WAFR)*, pp. 141–153, 1998.

[5] H. Kurniawati and D. Hsu, "Workspace importance sampling for probabilistic roadmap planning," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, vol. 2, pp. 1618–1623, Sept. 2004.

[6] J. P. van den Berg and M. H. Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners," *Int. J. Robot. Res.*, vol. 24, no. 12, pp. 1055–1071, 2005.

[7] E. Plaku, L. Kavraki, and M. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Trans. Robot.*, vol. 26, pp. 469–482, June 2010.

[8] J. Denny, *Collaborative Motion Planning*. PhD thesis, Texas A&M University, 2016.

[9] J. Denny, R. Sandström, A. Bregger, and N. M. Amato, "Dynamic region-biased exploring random trees," in *Alg. Found. Robot. XII*, Springer, 2020. (WAFR '16).

[10] R. Sandström, A. Bregger, B. Smith, S. Thomas, and N. M. Amato, "Topological nearest-neighbor filtering for sampling-based planners," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 3053–3060, 2018.

[11] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, pp. 566–580, Aug. 1996.

[12] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[13] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SCC-4, no. 2, pp. 100–107, 1968.

[14] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," tech. rep., Iowa State University, 1998.

[15] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 473–479, 1999.

[16] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 995–1001, 2000.

[17] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: an obstacle-based PRM for 3d workspaces," in *Proc. Int. Wksp. Alg. Found. Robot. (WAFR)*, (Natick, MA, USA), pp. 155–168, A. K. Peters, Ltd., 1998.

[18] J. Denny and N. M. Amato, "Toggle PRM: A coordinated mapping of C-free and C-obstacle in arbitrary dimension," in *Alg. Found. Robot. X*, pp. 297–312, Springer, 2013. (WAFR '12).

[19] L. Guibas, C. Holleman, and L. Kavraki, "A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, vol. 1, pp. 254–259, 1999.

[20] J.-M. Lien, S. Thomas, and N. Amato, "A general framework for sampling on the medial axis of the free space," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 3, pp. 4439–4444, Sept. 2003.

[21] G. Liu and J.-M. Lien, "Fast medial-axis approximation via Max-Margin pushing," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2015.

[22] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, pp. 846–894, 2011.

[23] Y. Li, Z. Littlefield, and K. E. Bekris, "Sparse methods for efficient asymptotically optimal kinodynamic planning," in *Alg. Found. Robot. XI*, pp. 263–282, Springer, 2015. (WAFR '14).

[24] J. Bialkowski, M. Otte, S. Karaman, and E. Frazzoli, "Efficient collision checking in sampling-based motion planning via safety certificates," *Int. J. Robot. Res.*, vol. 35, no. 7, pp. 767–796, 2016.

[25] M. Ghosh, S. Thomas, and N. M. Amato, "Fast collision detection for motion planning using shape primitive skeletons," in *Alg. Found. Robot. XIII*, Springer, 2020. (WAFR '18).

[26] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, pp. 509–517, Sept. 1975.

[27] A. Yershova and S. M. LaValle, "Improving motion-planning algorithms by efficient nearest-neighbor searching," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 151–157, 2007.

[28] J. Ichnowski and R. Alterovitz, "Fast nearest neighbor search in se(3) for sampling-based motion planning," in *Alg. Found. Robot. XI*, pp. 197–213, Springer, 2015. (WAFR '14).

[29] A. M. Kibriya and E. Frank, "An empirical comparison of exact nearest neighbor algorithms," in *Proc. of the Euro. Conf. Data Min. Know. Disc.*, 2007.

[30] J. Pan, C. Lauterbach, and D. Manocha, "Efficient nearest-neighbor computation for gpu-based motion planning," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2010.

[31] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala, "Locality-preserving hashing in multi-dimensional spaces," in *Proc. Annu. ACM Sympos. Theory Comput.*, pp. 618–625, 1997.

[32] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirronki, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. Annu. Symp. on Comput. Geom.*, pp. 253–262, 2004.

[33] E. Plaku and L. Kavraki, "Quantitative analysis of nearest-neighbors search in high-dimensional sampling-based motion planning," in *Alg. Found. Robot. VII*, pp. 3–18, Springer, 2006. (WAFR '06).

[34] S. Brin, "Near neighbor search in large metric spaces," in *VLDB J.*, pp. 574–584, 1995.

[35] H.-T. L. Chiang, A. Faust, S. Sugaya, and L. Tapia, "Fast swept volume estimation with deep learning," in *Alg. Found. Robot. XIII*, Springer, 2020. (WAFR '18).

[36] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*. Berlin, Germany: Springer-Verlag, 2nd ed., 2000.

[37] H. Si, "Tetgen, a delaunay-based quality tetrahedral mesh generator," *ACM Trans. Math. Softw.*, vol. 41, pp. 11:1–11:36, Feb. 2015.

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

[39] G. Reeb, "Sur les points singuliers d'une forme de pfaff complement integrable ou d'une fonction numerique," *Comptes Rendus Acad. Sciences Paris*, vol. 222, pp. 847–849, 1946.

[40] A. Tagliasacchi, I. Alhashim, M. Olson, and H. Zhang, "Mean curvature skeletons," in *Proc. Symp. Geom. Proc.*, vol. 31, pp. 1735–1744, 2012.

[41] H. Ling and D. W. Jacobs, "Shape classification using the inner-distance," *IEEE Trans. Patt. Analy. Mach. Intell.*, vol. 29, no. 2, pp. 286–299, 2007.

[42] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *Proc. IEEE Symp. Found. Comp. Sci. (FOCS)*, pp. 49–60, IEEE, 1987.

[43] Y.-S. Liu, K. Ramani, and M. Liu, "Computing the inner distances of volumetric models for articulated shape description with a visibility graph," *IEEE Trans. Patt. Analy. Mach. Intell.*, vol. 33, no. 12, pp. 2538–2544, 2011.

[44] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, pp. 378–400, May 2001.

[45] A. Yershova, L. Jaillet, T. Simeon, and S. M. Lavalle, "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 3856–3861, Apr. 2005.

[46] A. Hatcher, *Algebraic Topology*. Cambridge University Press, 2001.

[47] T. McMahon, S. L. Thomas, and N. M. Amato, "Reachable volume RRT," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, (Seattle, WA, USA), pp. 2977–2984, May 2015.

[48] A. A. Canutescu and R. L. Dunbrack, Jr., "Cyclic coordinate descent: A robotics algorithm for protein loop closure," *Protein Sci.*, vol. 12, no. 5, pp. 963–972, 2003.

[49] J. Denny, R. Sandström, N. Julian, and N. M. Amato, "A region-based strategy for collaborative roadmap construction," in *Alg. Found. Robot. XI*, pp. 125–141, Springer, 2015. (WAFR '14).

[50] L. Shapira, A. Shamir, and D. Cohen-Or, "Consistent mesh partitioning and skeletonisation using the shape diameter function," *The Visual Computer*, vol. 24, no. 4, pp. 249–259, 2008.

[51] K. Shi, J. Denny, and N. M. Amato, "Spark PRM: Using RRTs within PRMs to efficiently explore narrow passages," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, (Hong Kong, P. R. China), June 2014.

[52] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr, "On the design of CGAL a computational geometry algorithms library," *Softw. – Pract. Exp.*, vol. 30, no. 11, pp. 1167–1202, 2000.

[53] M. Eitz and G. Lixu, "Hierarchical spatial hashing for real-time collision detection," in *IEEE Int. Conf. Shape Model. Appl. (SMI)*, pp. 61–70, June 2007.