2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Spark PRM: Using RRTs Within PRMs to Efficiently Explore Narrow Passages

Kensen Shi, Jory Denny, and Nancy M. Amato

Abstract— **Probabilistic RoadMaps (PRMs) have been successful for many high-dimensional motion planning problems. However, they encounter difficulties when mapping narrow passages. While many PRM sampling methods have been proposed to increase the proportion of samples within narrow passages, such difficult planning areas still pose many challenges. We introduce a novel algorithm, Spark PRM, that sparks the growth of Rapidly-expanding Random Trees (RRTs) from narrow passage samples generated by a PRM. The RRT rapidly generates further narrow passage samples, ideally until the passage is fully mapped. After reaching a terminating condition, the tree stops growing and is added to the roadmap. Spark PRM is a general method that can be applied to all PRM variants. We study the benefits of Spark PRM with a variety of sampling strategies in a wide array of environments. We show significant speedups in computation time over RRT, Sampling-based Roadmap of Trees (SRT), and various PRM variants.**

I. INTRODUCTION

Motion planning is an important research area in robotics. Robots move to perform various tasks and thus need to compute feasible (e.g., collision-free) paths between start and goal configurations. When viewed more generally, motion planning has important applications outside robotics, such as in bioinformatics [23], computer animation [16], and computer aided design (CAD) [2]. Computing exact solutions to motion planning problems is intractable [21]. To overcome this, sampling-based methods [11], [15] have been used to solve many motion planning problems.

Two sampling-based methods, Probabilistic RoadMaps (PRMs) [11] and Rapidly-exploring Random Trees (RRTs) [15], have become widely used motion planning paradigms. PRM computes a roadmap by randomly generating configurations and using a local planner (e.g., straight-line) to connect nearby configurations. The roadmap can be queried for paths by connecting the start and goal configurations to the roadmap and performing a graph search (e.g., A^*). RRTs locally explore the planning space by expanding one or more trees in random directions until either a query can be solved or a computation limit is reached. At each expansion, a random configuration is generated and the closest node in the tree to that configuration is pushed in that direction to create a new node.

Both PRMs and RRTs encounter difficulties when planning in narrow passages, which often arise in high-dimensional problems or applications such as computer aided design [2] (e.g., removal of specific engine parts). Several approaches have been proposed to improve PRM [1], [3], [4], [8], [26] and RRT [22], [28] performance for such problems. Sampling-based Roadmap of Trees (SRT) [20] is a hybrid method that combines PRM and RRT methodologies by growing RRTs from configurations generated by PRM. A major motivation behind SRT is to efficiently exploit parallel processing.

In this paper, we present a novel algorithm, called Spark PRM, that uses PRM to quickly cover large areas of the planning space while using RRTs to locally explore narrow passages. In Spark PRM, a narrow passage test is performed on each node generated by an iterative PRM planner, and those within a narrow passage will spark the growth of an RRT. While the overall structure of Spark PRM is similar to SRT, the narrow passage test is the key feature that sets Spark PRM apart from SRT. In particular, instead of generating RRTs everywhere in the planning space, Spark PRM constrains RRT growth to narrow passages. RRTs are particularly suited for mapping narrow passages if they begin inside the passage (in such situations, the tree will be confined inside the passage and will thus explore it more efficiently). Spark PRM uses RRTs to rapidly generate further narrow passage configurations, therefore multiplying the beneficial effects of random narrow passage samples. In this way, Spark PRM combines the strengths of PRMs and RRTs, using each strategy where appropriate. The Spark PRM strategy can be applied to all PRM variants and can use any RRT variant (or other tree-based planner). The contributions of this paper include:

- Introducing Spark PRM, a novel combination of PRM and RRT for mapping spaces with narrow passages.
- Experimentally demonstrating the benefits of Spark PRM with various PRM techniques in a range of difficult planning problems with narrow passages. We show a significant reduction in computation time required for roadmap construction compared with RRT, SRT, and various PRM variants.

This research supported in part by NSF awards CNS-0551685, CCF-0833199, CCF-0830753, IIS-0916053, IIS-0917266, EFRI-1240483, RI-1217991, by NIH NCI R25 CA090301-11, by Chevron, IBM, Intel, Oracle/Sun and by Award KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST). J. Denny supported in part by an NSF Graduate Research Fellowship.

Kensen Shi, Jory Denny, and Nancy M. Amato are with the Parasol Lab, Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA, {kshi, jdenny, amato} at cse.tamu.edu. K. Shi graduated from A&M Consolidated High School, College Station, TX, in May 2013 and started undergraduate studies at Stanford University, Stanford, CA, in Sept. 2013.

II. RELATED WORK

In this section, we first present the basic preliminary definitions for motion planning. We then describe the most significant prior work relevant to the algorithm introduced, which is divided into three main groups: graph-based planners designed to improve mapping of narrow passages, tree-based planners for narrow passages, and hybrid methods combining graph-based and tree-based strategies.

A. Motion Planning Preliminaries

A robot is a movable object whose position and orientation can be described by n parameters, or *degrees of freedom* (DOFs), each corresponding to a positional variable (e.g., object positions, object orientations, link angles, and/or link displacements). Hence, a robot's placement, or configuration, can be uniquely described by a point $\langle x_1, x_2, ..., x_n \rangle$ in an n-dimensional space (where x_i is the ith DOF). This space, consisting of all possible robot configurations (valid or not), is called *configuration space* ($\mathcal{C}_{space}$) [18]. The union of all valid configurations is the *free space* ($\mathcal{C}_{free}$), while the union of the invalid configurations is the *blocked* or *obstacle space* ($\mathcal{C}_{obst}$). Thus, the motion planning problem becomes that of finding a continuous trajectory through $\mathcal{C}_{free}$ from a given start configuration to a goal configuration. In general, it is intractable to compute explicit $\mathcal{C}_{obst}$ boundaries [21], but we can often determine a configuration's validity quite efficiently, e.g., by performing a collision detection (CD) test in the *workspace*, the robot's natural space. Sampling-based planners (e.g., Probabilistic RoadMaps (PRMs) [11] and Rapidly-exploring Random Trees (RRTs) [15]) have had much success at solving diverse motion planning problems.

B. Graph-based Planners

PRMs [11] construct roadmaps of $\mathcal{C}_{free}$ by randomly sampling valid configurations and validating simple paths between nearby samples to form the edges of the roadmap. The roadmap is then queried for a final solution. However, PRMs are inefficient at mapping narrow passages [10] because it is difficult to generate samples inside the passages.

Many variants of PRMs aim to better map narrow passages. Obstacle-Based PRM (OBPRM) [1] and Uniform OBPRM [27] sample near $\mathcal{C}_{obst}$ surfaces. Gaussian PRM [3] and Bridge Test PRM [8] are filtering techniques with inexpensive tests aimed at either finding samples near $\mathcal{C}_{obst}$ boundaries or directly in narrow passages, respectively. Medial Axis PRM (MAPRM) [17], [26] pushes randomly sampled configurations to the medial axis. Toggle PRM [4] performs a coordinated mapping of both $\mathcal{C}_{free}$ and $\mathcal{C}_{obst}$, retaining witnesses from failed connection attempts in one space to augment the roadmap in the opposite space. These methods have their benefits and downsides, and some may be more suited for certain scenarios than others.

C. Tree-based Planners

Tree-based planners (e.g., RRTs [15] and Expansive-Spaces Trees (ESTs) [9]) are especially suited for single-query scenarios. RRTs [15] gradually explore $\mathcal{C}_{space}$ from some start configuration q_{root} by iteratively selecting a random direction q_{rand} followed by finding q_{near} (the nearest node in the tree to q_{rand}), and stepping q_{near} towards q_{rand} to create q_{new}, which is then added to the tree. More precisely, once q_{near} is found, an *Extend* operation [12] is performed on q_{near}, which iteratively takes steps at the environmental resolution towards q_{rand}, checking validity at each step until either a distance of Δq, the $\mathcal{C}_{obst}$ boundary, or q_{rand} is reached. RRTs have deficiencies in exploring narrow spaces of the environment, often colliding with obstacles and failing to discover the entrances to narrow passages.

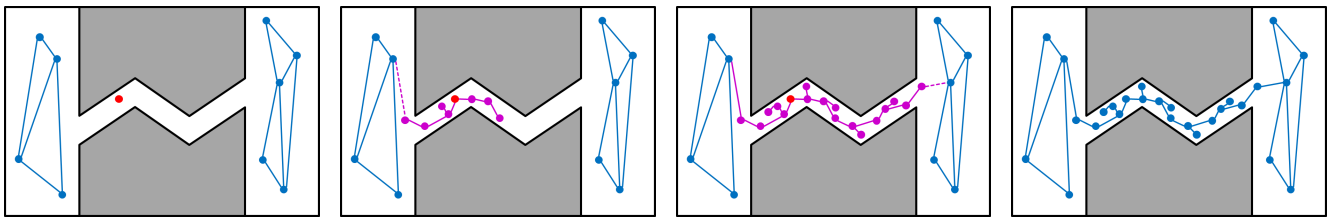
Many efforts have been made to improve RRTs. Along with introducing the expansion mechanism above, RRT-Connect [12] uses a bidirectional search with two trees, rooted at the start and goal. Obstacle-Based RRT (OBRRT) [22] biases expansion based on the workspace geometries of obstacles. Retraction-based methods [28] have had much success in mapping narrow passages by retracting nodes along obstacle boundaries. Adaptive RRT [5] adjusts to the space based on the visibility of the expanded node.

RRTLocTrees [24] uses an idea similar to the one presented in this paper. It uses RRT-Connect but also grows *local trees* rooted at random samples that were unable to connect to the other RRTs. RRTLocTrees attempts to connect every new sample to local trees with probability p_{grow}, which tunes the growth of the two global trees (rooted at the start and goal) relative to the local trees. Attempts to merge trees occur when the bounding box of a local tree has grown.

Although Spark PRM and RRTLocTrees seem similar, they differ significantly. First, RRTLocTrees uses an underlying RRT-Connect planner, while our algorithm uses a PRM. This implies that the two methods are designed for different purposes: RRTLocTrees is geared toward single-query scenarios, while our development is a multiple-query approach. Second, the two methods have different motivations for growing local trees. RRTLocTrees uses local trees to prevent the waste of random samples in difficult planning regions that the RRT cannot reach. In contrast, Spark PRM retains such samples anyway and instead uses trees because they are relatively better at exploring narrow passages if they are rooted in the passage. Finally, RRTLocTrees does not focus on local trees – in a sense, the trees all grow in parallel. Spark PRM, on the other hand, focuses computation on expanding trees until some termination condition is reached, in order to quickly improve the connectivity of the roadmap.

D. Hybrid Methods

Efforts have been made to combine PRMs and RRTs, notably Sampling-based Roadmap of Trees (SRT) [20]. This hybrid method constructs a roadmap where the nodes themselves are trees generated by tree-based planners, specifically RRT and EST. The tree roots are randomly selected in $\mathcal{C}_{free}$. Connections between these trees are aided by bi-directional RRTs or ESTs between nearby nodes of neighboring trees. SRT is designed for parallel computing and is more decoupled than PRM and RRT.



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

Fig. 1: Progression of Spark PRM in an example environment. (a) shows a roadmap (blue) constructed by the iterative PRM with one sample inside a narrow passage (red). In (b), an RRT (magenta) rooted at this node grows until it connects to the roadmap. (c) shows the RRT reaching the other exit of the narrow passage and again connecting to the roadmap. Finally, RRT growth is terminated and the tree is added to the roadmap, shown in (d).

While Spark PRM is similar to SRT in that both methods use PRMs and RRTs together, Spark PRM is designed for mapping narrow passages instead of for parallel systems. Instead of constructing trees from every random sample, as in SRT, Spark PRM limits tree construction to narrow passages, where they provide the most benefit. In this sense, Spark PRM is not a roadmap of trees, but is rather a PRM augmented with trees in narrow passages.

Multi-Modal-PRM [7] can also combine different planning strategies and has shown success in manipulation and legged locomotion applications.

III. SPARK PRM

In this section, we describe the motivation and algorithmic components of Spark PRM, which uses RRTs to explore narrow passages within a PRM planner.

A. Motivation

The motivation for Spark PRM lies heavily on the observation that RRTs growing inside narrow passages are forced to stay inside the narrow passage until they find an exit, regardless of how narrow or long the passage is. Spark PRM "sparks" the construction of RRTs in narrow passages rooted at PRM samples that pass a narrow passage test. Ideally, these RRTs grow inside narrow passages until they find exits that allow them to connect to the roadmap built by the PRM. Thus, RRT growth will lead to a continual and rapid generation of narrow passage samples until the passage has been fully mapped. In practice, it is difficult to balance this ideal with robustness in different environments. However, Spark PRM still achieves significant efficiency boosts, as shown in Section IV.

B. Algorithm

Algorithm 1 Spark PRM: Applying RRTs to PRM.

1: Roadmap $G = V = \emptyset, E = \emptyset$
2: **while** $!done$ **do**
3: $\quad G = G \cup \texttt{IncrementallyGrowPRM}()$
4: $\quad$ **for all** $\{s \in V : \texttt{InNarrowPassage}(s)\}$ **do**
5: $\quad\quad G = G \cup \texttt{ConstructRRT}(s)$

Algorithmically, Spark PRM (Algorithm 1) is a simple extension of PRM [11]. As in incremental PRM, Spark PRM alternates between sampling and connecting until the roadmap meets some terminating criteria (e.g., solving a query). However, after the connection step, Spark PRM performs a narrow passage test (line 4) for each new configuration s, and if s is inside a narrow passage, an RRT is constructed with s as the root (line 5). Upon termination of RRT growth, the tree is added to the roadmap.

Since the `IncrementallyGrowPRM()` step may use any PRM technique, e.g., Toggle PRM [4], the Spark PRM methodology is quite general. When using Toggle PRM, RRTs are only sparked from valid narrow passage configurations in the free roadmap and not for invalid configurations, since the ultimate objective is to improve the connectivity of the free roadmap.

The functions `InNarrowPassage()` and `ConstructRRT()` are quite general and could be implemented in a number of ways. We describe our specific implementations in Sections III-E and III-F.

C. Example

As an example, we will follow the execution of Spark PRM in a narrow passage environment shown in Figure 1. The white areas represent $\mathcal{C}_{free}$ and the gray areas represent $\mathcal{C}_{obst}$. Spark PRM begins by using an iterative PRM planner, which constructs a roadmap (blue) in Figure 1(a). The red node passes the narrow passage test, and an RRT (magenta) is sparked (Figure 1(b)). When the RRT expands and reaches an exit of the narrow passage, it connects to the roadmap (dotted magenta edge). In Figure 1(c), the RRT continues expansion and again connects to the roadmap. The RRT grows toward the other exit of the narrow passage due to an optimization that prevents further expansion outside the narrow passage near the first connection, explained more fully in Section III-F. In Figure 1(d), the RRT terminates expansion and is added to the roadmap. Finally, the iterative PRM resumes, possibly sparking other RRTs, until the map is satisfactory (e.g., can solve a query).

D. Discussion

As discussed above, the Spark PRM strategy yields several benefits. First, the PRM needs to generate significantly fewer

samples within narrow passages, since they are explored largely by the RRTs. Second, the discovery of one sample within a narrow passage will lead to the rapid generation of many more. Third, Spark PRM does not need to find the entrances to narrow passages from the "outside"; instead, it does the reverse (i.e., exploring passages from the inside to the outside), which is a much easier task for RRTs. These benefits are all illustrated in the example above. Note that in the example, Spark PRM is able to map the narrow passage after only one sample within the passage.

E. Narrow Passage Test

We can utilize information about connection attempts in the narrow passage test, `InNarrowPassage()`. If s is unable to connect to any of its neighbors (i.e., s has low/zero visibility [19]), or if it connects to a sufficiently small connected component (CC), then we assume s is in a narrow passage. In other words, `InNarrowPassage()` returns true if the size of the CC containing s is smaller than a set limit (always equal to 3 in our experiments). Note that if s cannot be connected to the roadmap, the size of its CC is 1. This narrow passage test is based on the assumption that the samples generated by a PRM are more likely to land in large free areas than in narrow passages. Further, the samples in free areas are more likely to connect successfully and create large CCs, while the few samples in narrow passages are unlikely to form equally large CCs. We only perform the narrow passage test after the roadmap contains a set number of nodes to allow time for large CCs to form. In our experiments, this number varies between 20 and 50, where the higher values are used for spaces with lower visibility. If reasonably set, this parameter does not largely affect efficiency. Using this narrow passage test has another benefit: it often prevents sparking multiple overlapping RRTs in the same area of $\mathcal{C}_{space}$ because samples in narrow passages often connect to a previously-generated RRT, thus failing the narrow passage test. This heuristic simply determines if s is in a difficult planning area (e.g., cluttered areas), not necessarily in a narrow passage. However, our experiments show that Spark PRM still performs well in environments with cluttered regions. Other more accurate narrow passage tests could also be used, but they might be computationally expensive.

F. Constructing an RRT

The general idea for the `ConstructRRT()` step is straightforward, shown in Algorithm 2. The RRT root is added to the tree, and the RRT is grown (as in [15]) until some terminating criteria is reached. Other RRT variants (e.g., OBRRT [22]) could be used for this step as well. Then, an optional post-processing step is applied (line 6) that trims the RRT, removing unnecessary nodes to keep the roadmap from becoming too large. We next describe each of these components in more detail.

Expanding the RRT. This behaves as in [15] and is described in Section II-C.

Algorithm 2 $ConstructRRT()$: Constructing an RRT across a narrow passage.

Input: Configuration q_{root}
Output: Configurations within the narrow passage

```
1: Tree T = ∅
2: T.AddNode(q_root)
3: while !TerminateExpansion(T) do
4:    ExpandRRT(T)
5:    AttemptConnectToRdmp(T)
6: TrimTree(T)
7: return T
```

Connecting to the Roadmap. The algorithm attempts to connect each new RRT node with its closest neighboring node that is outside the RRT. After the RRT successfully connects to the roadmap, connections are no longer attempted to the connected CC. In other words, we only try to connect the RRT to nodes that are not in the same CC as the tree.

Terminating Expansion. Once the RRT has grown to a given cutoff size, or when the RRT has connected to two distinct CCs of the roadmap, the RRT expansion stops. We found that 50 to 200 is generally a good cutoff size for RRTs, where larger values are more appropriate for cluttered spaces and those with long narrow passages. Our experimental setups all use values in this range. We stop expansion after connecting to two different CCs based on the observation that most narrow passages are like bridges or tunnels between two relatively large areas of free space, which we assume will be mapped by the PRM. When the RRT connects to two different CCs of the roadmap, it thus signifies that the RRT has fully explored the narrow passage from one end to the other, and further growth is unnecessary. It is possible that a small CC will exist within the narrow passage, so if the RRT connects to a sufficiently small CC (3 nodes or fewer in our tests, consistent with the narrow passage test), then that CC is not counted.

When Spark PRM is used to solve a query in which the start or goal configurations lie inside a narrow passage, RRTs are sparked from the start and goal if they pass the narrow passage test. The growth of these RRTs is terminated after they connect to only one CC in the roadmap because the objective is to connect the start and goal to more accessible areas in the planning space. It might seem that this reduces the algorithm to a standard RRT, but that is not always the case. If the environment contains more than one narrow passage between the start and goal, the large free areas of $\mathcal{C}_{space}$ would be mapped by the PRM while only the narrow passages are explored by the sparked RRTs.

Trimming the RRT. If the RRT connects to two CCs, then it improves the connectivity of the roadmap, which should be preserved throughout this step. However, the RRT also contains many nodes that do not improve the roadmap connectivity, and these may be removed. Our implementation uses a parameter $trimDepth$ (always set to 1 in our experiments) that determines how greedy the trimming should be. We compute the path through the RRT between the two

connected CCs, and we delete all RRT nodes that are more than $trimDepth$ edges away from a node on the path. An exception is the tree root q_{root}, which we never delete in order to preserve any theoretical benefits provided by the specific sampling strategy used. If desired, `TrimRRT()` can also trim RRTs that connect to only one roadmap CC, using a path between q_{root} and the node that connects to the roadmap CC. Using this option can prevent the roadmap from becoming too large too quickly but can also throw away possibly useful computation.

Optional optimizations. We also explored other optimizations that sometimes improve Spark PRM's efficiency.

We noticed that the RRTs would often reach one narrow passage entrance before the other and would waste time expanding outside the narrow passage while trying to explore the passage further. To address this, we label nodes that connect to other roadmap CCs as *connection* nodes and prevent the RRT from expanding near connection nodes (dependent on the delta distance for RRT growth). This effectively restricts the RRT growth to the narrow passage. However, this optimization would not be appropriate in cluttered environments.

Sometimes, an RRT connects to another roadmap CC after only one or two expansions. This most often occurs when the RRT root is in a corner or very near an obstacle, and it cannot connect to the roadmap because the configuration cannot rotate freely without collision – only when the configuration moves away from the obstacle will it be able to connect to the roadmap. Our narrow passage test can incorrectly spark an RRT from such a configuration. To avoid constructing unnecessary RRTs, we terminate RRT growth if the RRT connects to the roadmap in two or fewer expansions. This optimization would be less useful for point or spherical robots, and may also hinder efficiency in environments with short narrow passages such as doorways where an RRT should connect to the roadmap quickly anyway.

When the roadmap becomes too large, the `AttemptConnectToRdmp()` function (line 5 of Algorithm 2) is costly. Even though only one connection is attempted, finding the closest node in the roadmap requires many distance computations. This issue can be mitigated by only connecting to the CC whose centroid is the closest. This strategy may have undesirable effects in certain environments when the roadmap CC containing the closest node is different from the CC with the closest centroid.

IV. EXPERIMENTS

In this section, we experimentally analyze Spark PRM using several sampling techniques in a range of motion planning problems. We compare its effectiveness to that of the corresponding PRM planners, RRT, and SRT. We study Spark PRM's impact on many PRM variants and show that it is more efficient than each previous method tested.

A. Experimental Setup

The experiments were run on a Rocks Cluster running CentOS 5.1 with Intel XEON CPU 2.4 GHz processors with the GNU gcc compiler version 4.1. Each test was allotted a maximum of 10 hours of computation time before it was considered a failure.

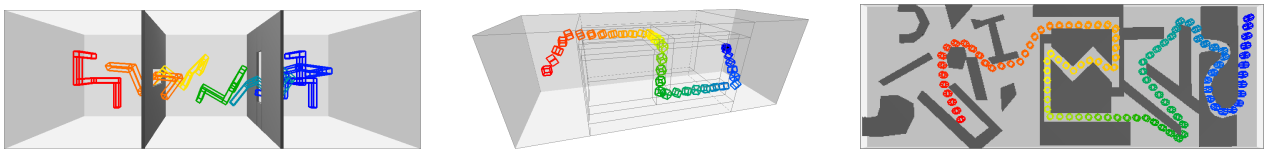
We tested Spark PRM with each of Gaussian PRM, OBPRM, Toggle PRM, and the basic PRM with uniform random sampling (referred to as Uniform PRM) as the base PRM planner. We will refer to these Spark PRM variations collectively as the Spark PRM methods. We compared each of the Spark PRM methods to the corresponding PRM planner (without the construction of any RRTs), which we refer to collectively as the PRM methods. For further comparison, we also tested RRT and SRT with uniform sampling.

All methods analyzed in this section were implemented in a C++ motion planning library developed in the Parasol Lab at Texas A&M University. This library uses a distributed graph data structure from the Standard Template Adaptive Parallel Library (STAPL) [25], a C++ library designed for parallel computing. However, experiments were all done sequentially, not in parallel. All methods use the Euclidean distance metric and a straight-line local planner in a k-closest connection strategy, where $k = 5$ for all PRM methods. All validity tests were collision detection (CD) tests using RAPID [6] or PQP [14]. For Gaussian sampling, the parameter d was manually chosen for each environment depending on the width of narrow passages. SRT parameters were generally based on recommendations in [20] with some tuning to increase efficiency.

We tested Spark PRM first with parameters chosen to suit each environment and sampler method, and then with a constant set of parameters to test the feasibility of an untuned Spark PRM for general-purpose use.

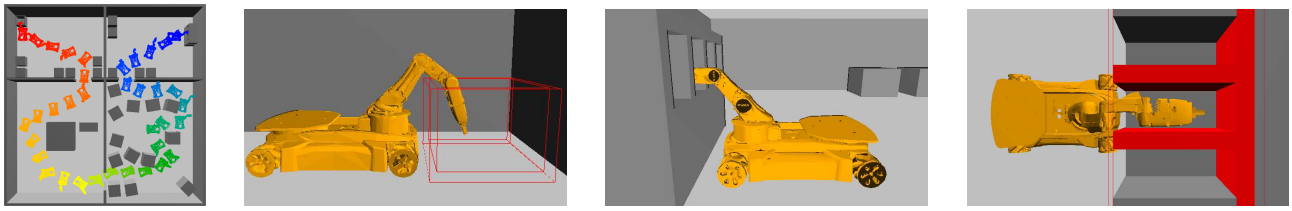
In the first round of tests, the RRT cutoff size is always between 50 and 200, with higher values used in environments with clutter or long narrow passages. The number of initial samples (before RRTs are sparked) is between 20 and 50, with higher values used in environments with lower visibility. These two parameters are the only two Spark PRM-specific parameters that differed across test cases (although the RRT parameters, such as the delta distance for expansion, are chosen for each environment). Our narrow passage test always uses 3 as the maximum size of a CC within a narrow passage, and the trimming parameter is always 1. In Section III-F, several optional optimizations were presented that sometimes help and other times hinder efficiency. Different combinations of these optimizations seem to work best depending mostly on the environment structure and slightly on the PRM variant used. We chose an appropriate combination of optimizations for each Spark PRM method.

Then, we repeated the tests while keeping the Spark PRM parameters constant (but the RRT-specific parameters still vary, because they can be easily selected based on the environment). We used 150 as the cutoff RRT size and 40 for the number of initial samples. As before, the maximum narrow passage CC size is 3, and the trimming parameter is 1. Here, we use only the first two of the optional optimizations described in Section III-F (using connection nodes and stopping RRT growth if the RRT connects to the



(a) Hook (b) ZTunnel (c) Heterogeneous

Fig. 2: Various environments for experimental analysis. All queries must traverse through narrow passages between start (red) and goal configurations (blue). Example paths are shown.



(a) YouBot (b) Start (c) Goal (d) Goal (top view)

Fig. 3: The YouBot environment. An example path is shown from start (red) to goal (blue). The start and goal configurations are also shown, with certain obstacles shown in a wire frame and/or red color for clarity.

roadmap too quickly).

For each setup tested, we report the success rate and the average time required to solve the query. Outliers (based on standard statistical analysis) are counted as completed test cases but are not factored into the average time. Results are computed from 50 trials per method.

Environments are shown in Figure 2 and 3(a), each with an example path from the start (red) to the goal (blue).

- In the Hook environment (Figure 2(a)), a 6DOF robot shaped like a hook must pass through two small openings in walls.
- In ZTunnel (Figure 2(c)), a 6DOF cube must traverse a long narrow passage that snakes through a block. Three different robot sizes are used to vary the narrow passage width. All ZTunnels have a narrow passage with a square cross section with side length 1.0. The cubical robot has a side length of 0.2 in ZTunnel1, 0.5 in ZTunnel2, and 0.6 in ZTunnel3.
- In Heterogeneous (Figure 2(d)), a 2DOF circular robot must travel through narrow passages and clutter.
- In YouBot (Figure 3(a)), an 8DOF KUKA youBot model begins a query when reaching into an open box (red wireframe in Figure 3(b)). It must then pass through doorways while navigating around boxes in an industrial scene. The query ends with the robot reaching into a cabinet on a table (Figure 3 (c)). A top view of the goal configuration, with the top shelf in a red wire frame and the vertical walls of the cabinet compartment in solid red, is shown in Figure 3(d).

The YouBot environment was created to simulate testing Spark PRM with a real robot. The KUKA youBot has an omnidirectional base and an arm with 5 joints, a total of 8DOF. The authors simplified the model obtained at [13], reducing the number of triangles to 19,780. The original youBot has a gripper on the arm, but the gripper is made immobile in our model and no gripping is actually performed.

B. Results

Table I shows the success rate and average time for each method tested in each environment. As noted above, we tested a tuned and untuned version of Spark PRM, in the table as "Spark PRM" and "Untuned", respectively. In instances where a "—" is listed as the average time, no average can be computed because no trials completed before the 10 hour computation limit. The speedups listed are computed by dividing the average time of the PRM, RRT, or SRT method by that of the corresponding tuned Spark PRM method (i.e., each speedup entry equals the number immediately above it divided by the average for the corresponding Spark PRM variation). For speedup calculations, entries of "—" are treated as 36,000 seconds. In all situations where Spark PRM is compared to a method that does not solve all trials, the true speedup is larger than listed in the table because failures are not included in the averages.

Hook. The two narrow passages are high in dimension, each requiring a complex series of rotations along with translational motion. All methods except Spark PRM had difficulties in Hook. The Spark PRM methods perform very well, solving the query several times faster than SRT and achieving speedups in the thousands compared to PRM.

ZTunnel. The three ZTunnel environments show varied results. Analyzing Spark PRM in ZTunnel gives important insights on Spark PRM's strengths.

In ZTunnel1, the easiest of the three, Spark PRM with Gaussian sampling is not much faster than Gaussian PRM itself. This occurs because Gaussian PRM creates many samples in the narrow passage, which is indeed not that narrow, reducing the growth of RRTs. In this case, Spark PRM's benefits diminish, but the overall strategy is not detrimental to the runtime, especially with our quick narrow passage test.

ZTunnel2 has a robot more than twice the size of the robot in ZTunnel1. Spark PRM boasts much higher speedups in

Runtimes (s) of Spark PRM with Various Sampling Methods, Compared to Corresponding Previous Methods												
Environment	Hook		ZTunnel1		ZTunnel2		ZTunnel3		Heterogeneous		YouBot	
Spark PRM (Uniform)	123.1	(100%)	8.119	(100%)	9.953	(100%)	33.81	(100%)	21.08	(100%)	1507	(100%)
Untuned (Uniform)	213.3	(100%)	14.08	(100%)	11.28	(100%)	39.61	(100%)	121.2	(100%)	3212	(100%)
Uniform PRM	—	(0%)	59.61	(100%)	2524	(100%)	—	(0%)	1621	(100%)	15600	(24%)
Speedup	**292.5**		**7.342**		**253.7**		**1065**		**76.91**		**10.35**	
RRT	10710	(90%)	40.66	(100%)	352.6	(100%)	773.7	(100%)	3785	(100%)	1396	(4%)
Speedup	**87.02**		**5.008**		**35.44**		**22.88**		**179.6**		**0.926**	
SRT	603.1	(100%)	27.68	(100%)	1887	(100%)	22060	(30%)	45.99	(100%)	6047	(100%)
Speedup	**4.899**		**3.409**		**189.6**		**652.5**		**2.182**		**4.013**	
Spark PRM (Gaussian)	22.53	(100%)	11.33	(100%)	5.983	(100%)	13.58	(100%)	13.75	(100%)	929.6	(100%)
Untuned (Gaussian)	22.75	(100%)	15.87	(100%)	6.883	(100%)	14.44	(100%)	20.07	(100%)	1226	(100%)
Gaussian PRM	—	(0%)	14.72	(100%)	49.48	(100%)	12690	(98%)	48.12	(100%)	15720	(44%)
Speedup	**1598**		**1.299**		**8.270**		**934.5**		**3.500**		**16.91**	
Spark PRM (OBRPM)	14.70	(100%)	5.990	(100%)	5.644	(100%)	17.10	(100%)	18.16	(100%)	1137	(100%)
Untuned (OBRPM)	19.03	(100%)	39.07	(100%)	7.367	(100%)	18.30	(100%)	37.23	(100%)	2252	(100%)
OBPRM	—	(0%)	63.59	(100%)	416.9	(100%)	24350	(100%)	79.41	(100%)	17610	(22%)
Speedup	**2449**		**10.62**		**73.86**		**1424**		**4.373**		**15.49**	
Spark PRM (Toggle)	58.40	(100%)	10.58	(100%)	8.311	(100%)	16.80	(100%)	15.05	(100%)	1082	(100%)
Untuned (Toggle)	80.60	(100%)	26.51	(100%)	12.93	(100%)	18.80	(100%)	26.61	(100%)	1454	(100%)
Toggle PRM	—	(0%)	107.7	(100%)	15460	(42%)	—	(0%)	76.96	(100%)	20442	(8%)
Speedup	**616.4**		**10.18**		**1861**		**2142**		**5.114**		**18.89**	

TABLE I: Runtimes in seconds of various methods in different environments. Success rate (out of 50 trials) is shown in parentheses. Trials that run over 10 hours are failures. The speedup provided by Spark PRM is also shown in bold.

ZTunnel2 than in ZTunnel1, and in general it maps ZTunnel2 *faster* than it maps ZTunnel1, even though ZTunnel2 is more difficult. This can be explained by the increased use of RRTs in ZTunnel2, since samples inside the narrow passage are more likely to spark RRTs due to failed connections. Note that Toggle PRM experiences relatively more difficulty in ZTunnel2 compared to the other PRM methods because the narrow passage is not α-ϵ-separable [4]. However, Spark PRM with an underlying Toggle PRM strategy does not suffer much from this issue.

ZTunnel3 is the most difficult ZTunnel variant, because the robot cannot fully rotate in the narrow passage. This explains the enormous jump in difficulty between ZTunnel2 and ZTunnel3. None of the previous methods are able to solve ZTunnel3 in a reasonable amount of time, but each Spark PRM method is able to do so quickly with speedups again in the thousands. Note that as the problem difficulty increases from ZTunnel1 to ZTunnel3, Spark PRM performance does not decay nearly as quickly as that of the previous methods.

ZTunnel2 and ZTunnel3 are examples of environments where Spark PRM will greatly outperform SRT, namely those with long narrow passages. Both Spark PRM and SRT choose roots of trees randomly with a sampler, leading to fewer trees in narrow passages than in easily-mapped spaces. SRT uses a fixed size for each RRT, which raises issues with long narrow passages: either the RRTs are large enough to expand through the passage but will dominate runtime by severely oversampling in the large free areas, or are too small to connect through the narrow passage with the few RRTs that are grown inside it. Spark PRM addresses this issue by limiting RRT growth to narrow passages, so a high RRT size limit would allow for the successful mapping of the passage with few RRTs yet will not hinder performance when mapping easy spaces.

Heterogeneous. The Heterogeneous environment shows that Spark PRM performs better than other methods even in cluttered environments. Note that our narrow passage test sometimes causes the growth of RRTs in cluttered areas, yet Spark PRM still boasts increased efficiency. It is also interesting to note that Heterogeneous is an environment particularly difficult for RRTs to solve because the solution path winds through the environment and at times requires q_{rand} to be selected from a small region in order to expand the tree in the correct direction. However, Spark PRM uses many RRTs, so this problem is much less pronounced. We note that in 2D, problems such as Heterogeneous can be solved with exact geometric analysis, but we use this example to compare the methods in a wider array of problems, not to provide a faster solution to 2D problems.

YouBot. In the YouBot environment, Spark PRM again outperforms the other methods, running 4 times faster than SRT, which is the only other method that consistently solves the query. The PRM variations and RRT all have success rates lower than 50%. The complexity of the youBot model makes collision detection time-consuming. Using newer computer hardware and a convex hull approximation of each robot section would shorten runtimes. However, since we only wish to compare Spark PRM's efficiency to that of previous methods, obtaining short runtimes was not a main concern. Note that the average time used by RRT as listed in the table is smaller than that for Spark PRM, giving a speedup smaller than 1. However, RRT only solved 2 out of 50 trials (failed trials are not included in the average time), while Spark PRM solved all 50. Spark PRM's success in YouBot demonstrates that Spark PRM is applicable to linkages and articulated robots.

Untuned Parameters. The untuned versions of Spark PRM are most often between 1 and 2 times slower than the tuned versions, and in a couple of exceptional cases about 6 times slower. In general, untuned Spark PRM still achieves speedups with the same order of magnitude. These results show that Spark PRM does not need to be carefully tailored to suit each different application; a single set of parameters will still perform well across many environments.

Overall, Spark PRM was able to solve 100% of test cases in each environment, unlike any of the other methods tested. In most situations, Spark PRM outperforms the other methods, sometimes achieving speedups of over 1000 times.

V. CONCLUSION

In this paper, we present a novel algorithm, Spark PRM, that uses RRTs in narrow passages to assist a PRM planner. Our new approach allows a single random sample inside a narrow passage to spark the continual and rapid generation of many more, thus efficiently mapping narrow passages. We experimentally analyzed Spark PRM with many sampling strategies in a diverse set of environments, demonstrating great speedups in computation time compared to previous methods. In the future, we would like to explore Spark PRM with a wider variety of narrow passage tests and tree planning techniques (including RRT variants and other tree-based planners, e.g. EST). A parallelized version of Spark PRM might also be interesting to explore.

REFERENCES

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: an obstacle-based PRM for 3d workspaces. In *Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective: the algorithmic perspective*, WAFR '98, pages 155–168, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[2] O. B. Bayazit, G. Song, and N. M. Amato. Enhancing randomized motion planners: Exploring with haptic hints. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 529–536, 2000.

[3] V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 1018–1023, May 1999.

[4] J. Denny and N. M. Amato. Toggle PRM: A coordinated mapping of C-free and C-obstacle in arbitrary dimension. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, Cambridge, Massachusetts, USA, June 2012.

[5] J. Denny, M. Morales, S. Rodriguez, and N. M. Amato. Adapting RRT growth for heterogeneous environments. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, November 2013. To appear.

[6] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. *Comput. Graph.*, 30:171–180, 1996. Proc. SIGGRAPH '96.

[7] K. Hauser and J.-C. Latombe. Multi-modal motion planning in non-expansive spaces. *Int. J. Robot. Res.*, 29(7):897–915, 2010.

[8] D. Hsu, T. Jiang, J. Reif, and Z. Sun. Bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4420–4426, 2003.

[9] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *Int. J. Robot. Res.*, 21(3):233–255, March 2002.

[10] D. Hsu, J.-C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Int. J. Robot. Res.*, 25:627–643, July 2006.

[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.*, 12(4):566–580, August 1996.

[12] 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)*, pages 995–1001, 2000.

[13] KUKA. http://www.youbot-store.com.

[14] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Distance queries with rectangular swept sphere volumes. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 4, pages 3719–3726 vol.4, 2000.

[15] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Int. J. Robot. Res.*, 20(5):378–400, May 2001.

[16] J.-M. Lien, O. B. Bayazit, R.-T. Sowell, S. Rodriguez, and N. M. Amato. Shepherding behaviors. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4159–4164, April 2004.

[17] J.-M. Lien, S. Thomas, and N. Amato. A general framework for sampling on the medial axis of the free space. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 4439 – 4444, sept. 2003.

[18] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[19] M. A. Morales A., R. Pearce, and N. M. Amato. Metrics for analyzing the evolution of C-Space models. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1268–1273, May 2006.

[20] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Trans. Robot. Automat.*, 2005.

[21] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 421–427, San Juan, Puerto Rico, October 1979.

[22] S. Rodriguez, X. Tang, J.-M. Lien, and N. M. Amato. An obstacle-based rapidly-exploring random tree. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2006.

[23] G. Song and N. M. Amato. Using motion planning to study protein folding pathways. In *Proc. Int. Conf. Comput. Molecular Biology (RECOMB)*, pages 287–296, 2001.

[24] M. Strandberg. Augmenting RRT-planners with local trees. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 4, pages 3258–3262, 2004.

[25] G. Tanase, A. Buss, A. Fidel, Harshvardhan, I. Papadopoulos, O. Pearce, T. Smith, N. Thomas, X. Xu, N. Mourad, J. Vu, M. Bianco, N. M. Amato, and L. Rauchwerger. The STAPL Parallel Container Framework. In *Proc. ACM SIGPLAN Symp. Prin. Prac. Par. Prog. (PPoPP)*, pages 235–246, San Antonio, Texas, USA, 2011.

[26] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 1024–1031, 1999.

[27] H.-Y. C. Yeh, S. Thomas, D. Eppstein, and N. M. Amato. UOBPRM: A uniformly distributed obstacle-based PRM. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 2655–2662, Vilamoura, Algarve, Portugal, 2012.

[28] L. Zhang and D. Manocha. An efficient retraction-based RRT planner. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2008.