Lazy Toggle PRM: A Single-Query Approach to Motion Planning

Jory Denny, Kensen Shi, and Nancy M. Amato

Abstract— **Probabilistic RoadMaps (PRMs) are quite successful in solving complex and high-dimensional motion planning problems. While particularly suited for multiple-query scenarios and expansive spaces, they lack efficiency in both solving single-query scenarios and mapping narrow spaces. Two PRM variants separately tackle these gaps. Lazy PRM reduces the computational cost of roadmap construction for single-query scenarios by delaying roadmap validation until query time. Toggle PRM is well suited for mapping narrow spaces by mapping both $\mathcal{C}_{free}$ and $\mathcal{C}_{obst}$, which gives certain theoretical benefits. However, fully validating the two resulting roadmaps can be costly. We present a strategy, Lazy Toggle PRM, for integrating these two approaches into a method which is both suited for narrow passages and efficient single-query calculations. This simultaneously addresses two challenges of PRMs. Like Lazy PRM, Lazy Toggle PRM delays validation of roadmaps until query time, but if no path is found, the algorithm augments the roadmap using the Toggle PRM methodology. We demonstrate the effectiveness of Lazy Toggle PRM in a wide range of scenarios, including those with narrow passages and high descriptive complexity (e.g., those described by many triangles), concluding that it is more effective than existing methods in solving difficult queries.**

I. INTRODUCTION

Motion planning entails finding a valid (e.g., collision free) path for a movable object (e.g., robot) between a start and a goal configuration. Motion planning has broad application not only in robotics but also in areas such as computational biology [20] and computer animation [14]. Unfortunately, computing exact solutions is intractable [18]. Sampling-based methods [12], [13] successfully provide a feasible approach to solving problems in high-dimensional spaces. One approach, Probabilistic RoadMaps (PRMs) [12], computes a graph representation of the planning space. After sampling and adding valid configurations to the roadmap, a connection phase validates simple deterministic paths between neighboring configurations with a local planner (e.g., straight-line). Successful paths are stored as edges in the roadmap. Once a roadmap is constructed, it can be queried for a path by connecting both the start and goal configurations to the roadmap and performing a simple graph search (e.g., A^*). The resulting path through the graph is a feasible path between the given start and goal configurations.

This research supported in part by NSF awards CNS-0551685, CCF-0833199, CCF-0830753, IIS-0917266, IIS-0916053, EFRI-1240483, RI-1217991, by NIH NCI R25 CA090301-11, by THECB NHARP award 000512-0097-2009, 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, K. Shi, and N. M. Amato are with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX, 77843, USA {jdenny, kshi, amato} at cse.tamu.edu. K. Shi is a senior at A&M Consolidated High School, College Station, TX.

Even though PRMs have shown great success in mapping complex high-dimensional problems, they typically require many collision detection calls to validate the vertices and edges of the roadmap. This operation can be quite expensive in many scenarios. Additionally, PRMs are inefficient when mapping narrow passages [11]. Narrow passages can be found in many real world problems such as computer-aided design applications, e.g., testing the possibility of removing engine parts without moving any other parts [2].

There has been extensive work done to improve the effectiveness of PRMs and reduce their computational cost. Lazy strategies, such as Lazy PRM [3], Fuzzy PRM [17], and Customizable PRM [21], aim to reduce the computational cost of PRMs by delaying evaluation of vertices and edges of the roadmap until validation is absolutely necessary, i.e., when querying for a path. During roadmap construction, vertices and edges are assumed to be valid. Only when a path is extracted from the roadmap do expensive collision detection tests occur — first to validate the vertices and then to validate the edges composing a path. If the path is invalidated, the invalid vertices and edges are removed from the roadmap. By using lazy evaluation, Lazy PRM successfully reduces the cost of planning with PRMs for single-query scenarios.

Toggle PRM [7], [9] is a recently proposed method that maps both free space and obstacle space in a coordinated manner. This method relies on the phenomenon that roadmap construction in one space can be used to augment the roadmap in the other space. Toggle PRM accomplishes this by retaining witnesses to failed connection attempts, which in turn are added and connected in the opposite space's roadmap. Toggle PRM both theoretically and experimentally improves the efficiency of mapping narrow spaces.

In this paper, we present a strategy, Lazy Toggle PRM, which utilizes both lazy evaluation and the Toggle PRM methodology. Lazy Toggle PRM begins with a lazily constructed roadmap and iteratively extracts and validates paths from the lazy roadmap. The invalid portions of the roadmap naturally provide an obstacle roadmap used by Toggle PRM. If no path succeeds, the techniques from Toggle PRM augment the free map to target difficult areas of the planning space, e.g., narrow passages. In this way, Lazy Toggle PRM successfully employs both lazy evaluation and proven vertex enhancement techniques in a coordinated fashion, thus combining the strengths of both Lazy PRM and Toggle PRM. Specific contributions of this paper include:

- Introducing Lazy Toggle PRM which utilizes both lazy evaluation and techniques from Toggle PRM to efficiently solve single-query scenarios.

- Experimentally analyzing the effectiveness of Lazy Toggle PRM in a range of difficult planning problems, showing a reduction in computation time and collision detection calls required for roadmap construction.

Section II discusses other work related to this research, Section III describes the Lazy Toggle PRM algorithm, and Section IV provides an experimental analysis of the algorithm.

II. RELATED WORK

In this section, we describe basic preliminary definitions for motion planning and the most significant related work relevant to the algorithm introduced. These are divided into two main groups: PRM variants aiming to improve mapping of narrow passages and those aiming to reduce validity tests (e.g., collision detection tests).

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 an object component (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 $(x_1, x_2, ..., x_n)$ in an n-dimensional space (where x_i is the ith DOF). This space, consisting of all possible robot configurations (feasible or not), is called *configuration space* ($\mathcal{C}_{space}$) [16]. The subset of all feasible configurations is the *free space* ($\mathcal{C}_{free}$), while the union of the infeasible configurations is the *blocked* or *obstacle space* ($\mathcal{C}_{obst}$). Thus, the motion planning problem becomes that of finding a continuous trajectory in $\mathcal{C}_{free}$ from a given start configuration to a goal configuration. In general, it is intractable to compute explicit $\mathcal{C}_{obst}$ boundaries [18], but we can often determine whether a configuration is feasible or not quite efficiently, e.g., by performing a collision detection (CD) test in the *workspace*, the robot's natural space.

Sampling-based methods [12], [13] are quite successful at solving motion planning problems. Specifically, Probabilistic RoadMaps (PRMs) [12] map $\mathcal{C}_{free}$ by randomly sampling valid configurations and validating simple straight-line 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 [11].

PRM variants for mapping narrow passages. In order to improve the mapping of narrow passages, some variants attempt to map $\mathcal{C}_{obst}$ surfaces. Obstacle-Based PRM (OBPRM) [1] and Uniform OBPRM [24] sample configurations near $\mathcal{C}_{obst}$ surfaces either by pushing configurations to the $\mathcal{C}_{obst}$ boundary or by finding surface intersections of randomly placed line segments. Both methods can be expensive as they require many validity tests. Gaussian PRM [4] and Bridge Test PRM [10] 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) [15], [23] pushes randomly sampled configurations to the medial axis. It provably improves the probability of sampling in most narrow passages over uniform random sampling, but is computationally intensive even when approximations are used. All of these methods have their own benefits and downsides.

To efficiently map narrow passages, we adapt a strategy called Toggle PRM [7], [9]. This strategy performs a coordinated mapping of both $\mathcal{C}_{free}$ and $\mathcal{C}_{obst}$. It retains witnesses from failed connection attempts in one space to augment the roadmap in the opposite space. It theoretically and experimentally outperforms uniform random sampling and most of the other methods described above in many scenarios. Additionally, [9] provides a novel classification of narrow passages that can be solved efficiently by this methodology. Toggle Local Planning [8] uses these ideas to do local planning in a 2-dimensional triangular subspace of $\mathcal{C}_{space}$, which results in improved connectivity over the traditional straight-line local planner [12].

PRM variants for reducing validity tests. In many situations, checking validity of configurations is the most expensive operation for a PRM planner. Lazy PRM [3], [17], [21] remedies this by delaying validity tests as long as possible. Vertices and edges in a roadmap are initially assumed to be valid and are only validated when a potential path is found. If any vertex or edge in the path is found to be invalid, it is removed from the roadmap and a new path is extracted. If no more paths exist, more configurations are lazily added to the roadmap. Lazy PRM thus does not need to validate the entire roadmap; by only validating the portions relevant to a specific query, Lazy PRM improves efficiency. However, to reduce the number of validity tests, Lazy PRM uses more memory and graph operations than traditional PRM approaches. Fuzzy PRM [17] performs a two-level construction of roadmaps associating edges with their corresponding probabilities of collision. The path with the lowest probability of being invalid is extracted and incrementally validated until a completely valid path is found. Customizable PRM [21] abstracts these by allowing users to individually set validity requirements of vertices, edges, and paths, e.g., vertices are collision free, edges are lazily constructed, and paths require a minimum obstacle clearance.

Other efforts, such as [5], [6], seek to model $\mathcal{C}_{space}$ to more efficiently map $\mathcal{C}_{free}$. These approximate models use a predictor to reduce the number of validity tests. Furthermore, some such as [19] employ the benefits of clearance balls where a configuration of the roadmap represents a hypersphere of valid configurations. Spheres can filter and guide further sampling. Additionally, if the spheres overlap, an edge can be created with no extra cost, so both sampling and edge checking computations are reduced.

III. LAZY TOGGLE PRM

In this section, we describe the design, motivation, and benefits of the algorithmic components of Lazy Toggle PRM, shown in Algorithm 1. Lazy Toggle PRM deviates from other methods by combining two strategies: lazy evaluation and mapping both $\mathcal{C}_{free}$ and $\mathcal{C}_{obst}$ by utilizing witnesses from failed path validations or edge connections.

The algorithm initializes a free roadmap G_{free}, an obstacle roadmap G_{obst}, and a queue Q to be empty. Q will be used to store witnesses to failed path validation or failed vertex connection attempts. After the initialization of variables, Lazy Toggle PRM iterates over three phases: lazy roadmap construction (lines 5-7), path validation (lines 8-18), and witness processing (lines 19-27). The first phase uses lazy evaluation during standard roadmap construction until at least one path exists in the roadmap. At this time, the second phase will iteratively extract and validate portions of the graph until either a path is found or no path exists. If no path exists, the third phase employs techniques of Toggle PRM to effectively augment the roadmap. These three phases are repeated until a successful path is found or a maximum number of iterations is reached (implying no path is found).

Lazy roadmap construction. This phase samples configurations and lazily adds them to G_{free} without checking their validity until the start and goal are in the same connected component (i.e., connected by a path). Note that any level of laziness can be requested here, e.g., the sampling could validate configurations before adding them to the roadmap and only use lazy edge validation [21].

Path validation. The path validation phase begins when the start and goal are in the same connected component. While a path exists from the start to goal, this phase finds the shortest path (e.g., using A^*) and evaluates its validity. Vertices are validated before edges, and as in Lazy PRM [3] they are validated outside-in, beginning with portions closest to the start and goal and working towards the middle of the path. This ordering is used because vertices and edges closer to the start or goal are more likely to appear repeatedly in extracted paths, and thus their validity becomes more important. Since invalid edges are often detected quickly by coarse validations, edges are validated in a multi-resolution fashion, i.e., a fast but coarse validation is done for the entire path, followed by incrementally slower and finer validations. The resolutions can be user-defined. As soon as the algorithm finds an invalid vertex or edge, it is deleted from G_{free}, its witness (the vertex itself or an invalid configuration along the edge) is added to Q, and the path validation phase is repeated. If all vertices and edges are valid, then the entire path is valid, so the algorithm returns the path and terminates. When there are no more paths from the start to goal, the witness processing phase begins.

Witness processing. This phase handles witnesses stored in Q. The algorithm considers two cases, depending on the validity of the witness n extracted from Q. In the first case (n is invalid), n is added to G_{obst} and connected in G_{obst} in a non-lazy manner using ToggleConnect [9], which connects a vertex in a map to successive connected components until a failure is reached. Any witnesses to failed edge connections will be in $\mathcal{C}_{free}$ and enqueued in Q. The obstacle roadmap is constructed in a non-lazy manner for several reasons. Lazy evaluation of the vertices of G_{obst} is not necessary because their origin as witnesses implies that their validity is already known. Since only the parts of G_{free} relevant to the query are validated, G_{obst} only spans areas relevant to the query as well. Finally, because an important goal of Lazy Toggle PRM is to find samples in narrow passages, which are most often witnesses to edge validations in G_{obst}, such edge validations should not be postponed. In the second case (the witness n is valid), n is added to G_{free} and connected lazily, i.e., not validated. When Q is empty or a potential path is created, the algorithm returns to the lazy roadmap construction phase.

Algorithm 1 Lazy Toggle PRM Algorithm

```
Input: Query $(q_s, q_g)$
Output: Valid Path $P$
 1: Roadmaps $G_{free}, G_{obst} = \emptyset$
 2: Queue $Q = \emptyset$
 3: $G_{free}.AddVertices(\{q_s, q_g\})$
 4: while $\neg done$ do
      /* Lazy Roadmap Construction */
 5:   while $\neg IsSameCC(q_s, q_g)$ do
 6:     $G_{free}.AddVertices(LazySample())$
 7:     $LazyConnect(G_{free})$
      /* Path Validation */
 8:   while $IsSameCC(q_s, q_g)$ do
 9:     $P \leftarrow G_{free}.Path(q_s, q_g)$
10:     if $\neg ValidateVertices(P)$ then
11:       $Q.Enqueue(Witness(P))$
12:       $G_{free}.DeleteVertex(Witness(P))$
13:     else /* all vertices are valid */
14:       if $\neg ValidateEdges(P)$ then
15:         $Q.Enqueue(Witness(P))$
16:         $G_{free}.DeleteEdge(Witness(P))$
17:       else /* all edges are valid */
18:         Return $P$
      /* Witness Processing */
19:   while $\neg Q.IsEmpty() \wedge \neg IsSameCC(q_s, q_g)$ do
20:     $n \leftarrow Q.Dequeue()$
21:     if $n$ is invalid then
22:       $G_{obst}.AddVertex(n)$
23:       Witnesses $W \leftarrow ToggleConnect(G_{obst})$
24:       $Q.Enqueue(W)$
25:     else /* $n$ is valid */
26:       $G_{free}.AddVertex(n)$
27:       $LazyConnect(G_{free})$
```

Example. As an example, take the simple environment shown in Figure 1(a). The black areas represent $\mathcal{C}_{obst}$ and the white areas represent $\mathcal{C}_{free}$. The start and goal configurations are denoted by an S and G, respectively. After initialization, a lazy roadmap (gray) is constructed (Figure 1(b)). Then, the path validation phase of the algorithm begins extracting and validating paths (bold in Figure 1(c)). Fully validated portions of the roadmap are shown in blue, while unvalidated portions remain gray. Witnesses are shown in red, which will be added to G_{obst} during the witness processing phase. The path validation phase repeatedly removes invalid vertices and edges from the roadmap until no more paths are left (Figure 1(d)). From here, Q is processed, adding vertices to G_{obst} (red) and augmenting G_{free} because a failed edge

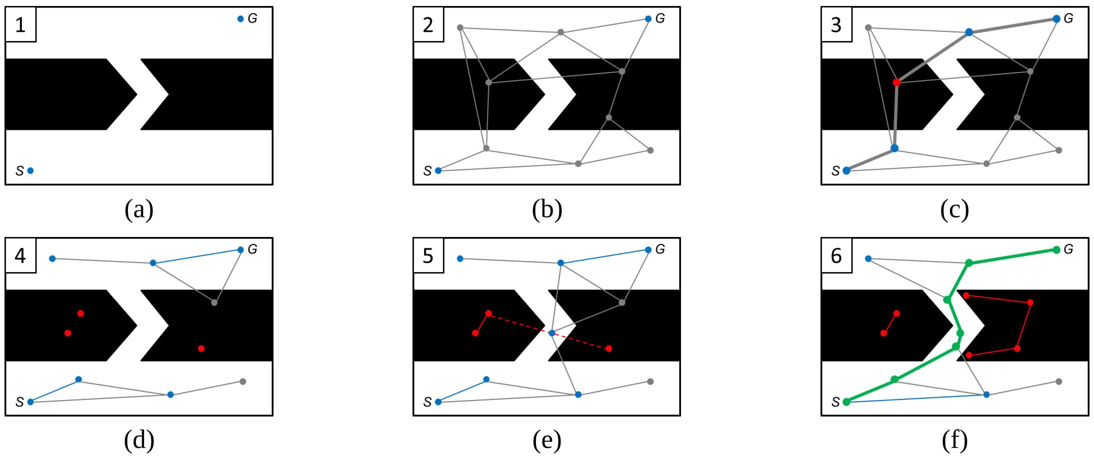


Fig. 1: Progression of Lazy Toggle PRM in a simple environment (a). (b) shows a lazily constructed roadmap (gray). (c) shows an extracted path (bold) with an invalid vertex (red); validated portions of the graph are shown in blue. (d) shows the roadmap after repeatedly removing invalid portions of the roadmap until no more paths are left (witnesses are shown in red). (e) presents roadmap augmentation through the strategies of Toggle PRM. After a few iterations, (f) shows the final roadmaps and extracted path (bold green).

validation in G_{obst} has yielded a witness in the narrow passage, as shown in Figure 1(e). This witness has been added to G_{free} and connected lazily. After a few iterations, a completely valid path is found (bold green in Figure 1(f)). Note that G_{free} is not guaranteed to lie completely in $\mathcal{C}_{free}$ due to lazy evaluation.

Discussion. This approach retains benefits from both methodologies. It reduces the computational cost of validating the roadmaps with lazy evaluation and efficiently augments the roadmaps using techniques from Toggle PRM.

First, utilizing lazy evaluation during roadmap construction eliminates the often unnecessary need to validate the entire roadmap. By delaying validation until query-time, the algorithm only validates what is needed for that particular query. After successfully completing a single query, the same algorithm can use the partially validated map for subsequent queries. Hence, even for multiple queries, lazy evaluation does not cause an increase in work; computations are simply postponed until needed.

Second, Toggle PRM theoretically provides for more efficient mapping of a specific class of narrow passages, namely α-ϵ-separable narrow passages, compared to uniform random sampling [9]. α-ϵ-separable narrow passages are narrow regions of $\mathcal{C}_{free}$ that locally separate components of $\mathcal{C}_{obst}$, e.g., a doorway partially separating sections of a building wall. By retaining witnesses in $\mathcal{C}_{obst}$ during path validation and in $\mathcal{C}_{free}$ from obstacle roadmap connection, Lazy Toggle PRM preserves the full benefits of Toggle PRM.

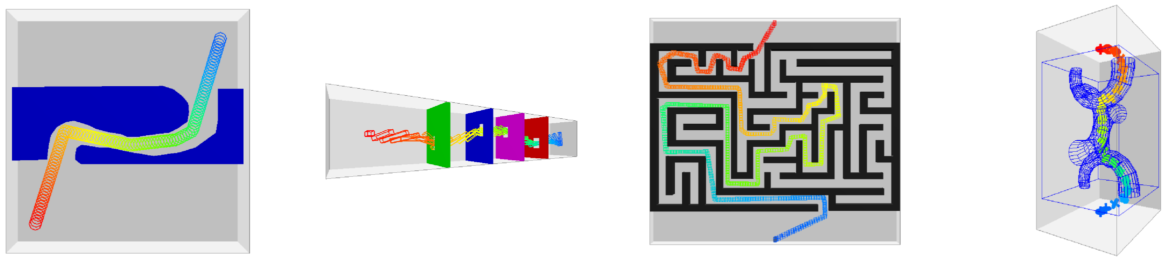
IV. EXPERIMENTS

In this section, we analyze Lazy Toggle PRM in a range of single-query problems, comparing its effectiveness to that of Lazy PRM, Toggle PRM, and Basic PRM. Section IV-A discusses the experimental setup, Section IV-B compares the methods in a few basic environments showing efficiency improvements for Lazy Toggle PRM, and Section IV-C compares the methods in a few complex scenarios.

A. Experimental Setup

All 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 1GB memory usage before considered a failure.

Basic PRM (referred to as Basic), Toggle PRM, Lazy PRM, and Lazy Toggle PRM 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) [22], a C++ library designed for parallel computing. All lazy methods (both Lazy PRM and Lazy Toggle PRM) use multi-resolution edge validation. An important parameter for the Lazy PRM and Lazy Toggle PRM methods is the degree of laziness for vertex validation, i.e., whether the free roadmaps are constructed from random samples of $\mathcal{C}_{space}$ or from $\mathcal{C}_{free}$ exclusively. We attempt to analyze the effects of varying levels of laziness in our experiments. The methods LazyFree and LazyToggleFree obtain samples from only $\mathcal{C}_{free}$, meaning vertices are validated before they are added to the roadmap. Both Lazy and LazyToggle obtain samples randomly distributed throughout $\mathcal{C}_{space}$. LazyToggleMix obtains 80% of its samples from $\mathcal{C}_{free}$ and 20% from $\mathcal{C}_{space}$. The intuition behind including LazyToggleMix is that Toggle PRM does well when seeded with a few configurations in $\mathcal{C}_{obst}$. We will refer to Lazy and LazyFree collectively as the Lazy PRM methods, and LazyToggle, LazyToggleFree, and LazyToggleMix collectively as the Lazy Toggle PRM methods. All methods utilize uniform random sampling in the initial roadmap construction phases. All methods use the Euclidean distance metric and straight-line local planner in a k-closest connection strategy. All methods used $k = 5$ except Uniform, which used $k = 10$.


(a) S2D (b) Walls (c) 2DMaze (d) MazeTunnel

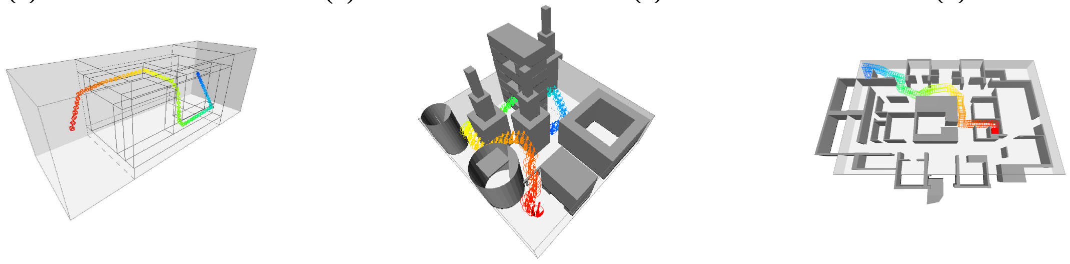

(e) ZTunnel (f) Helicopter (g) Building

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

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

- In S2D (Figure 2(a)), a simple 2DOF robot must travel through a relatively simple narrow passage between the two free spaces above and below the passage.
- In Walls (Figure 2(b)), a simple stick-like 6DOF robot must traverse a series of narrow passages (walls with holes) to find a path from end to end of the environment.
- In the 2DMaze (Figure 2(c)), a 2DOF robot must traverse a maze-like narrow passage. The winding passages make connection difficult for PRM methods.
- In the MazeTunnel (Figure 2(d)), a top-like 6DOF robot must traverse a long narrow passage with dead ends to solve a query from the top to bottom of the environment.
- In ZTunnel (Figure 2(e)), a 6DOF cube must traverse a long narrow passage within a block.
- In Helicopter (Figure 2(f)), a 6DOF helicopter must maneuver through a cityscape while avoiding buildings and cables. This environment has the constraint that the helicopter cannot rise above a certain elevation. Kinodynamic constraints are not considered.
- In Building (Figure 2(g)), a rectangular 3DOF robot (much like a refrigerator) must be moved outside the building while passing through very narrow doorways.

In our experiments, all validity tests were collision detection (CD) tests (i.e., “valid” is equivalent to “collision free” in our experiments). We report the number of successful completions, the number of CD calls needed to solve the query, the number of graph searches performed during the query phase (which become a significant portion of the computation time in lazy methods in the very simple environments), and the time needed to solve the queries (which measures the overall efficiency of each planner). The basic scenarios were run with 30 different seeds for random number generation, and the complex scenarios were run with 60 different seeds. The metrics reported are an average of the successful runs with outliers removed based on standard statistical analysis (the number of successes still includes these outlying runs).

B. Basic Scenarios

We analyzed both the effectiveness and trade-offs of the various methods in S2D, Walls, 2DMaze, MazeTunnel, and ZTunnel environments. The number of successful trials out of 30 is shown in Table I, a log scale of the number of total required CD calls normalized to Toggle is shown in Figure 3, a log scale of the number of total graph search operations normalized to LazyFree is shown in Figure 4, and a log scale of the total time required to solve the query normalized to Toggle is shown in Figure 5. All unsuccessful trials are due to excessive memory use (over 1GB in our tests).

Figure 3 shows a log scale of the number of CD calls required to solve the query normalized to Toggle. As seen in [3], lazy evaluation clearly reduced the number of CD calls required compared to Basic. All of the Lazy Toggle PRM methods replicate this benefit from their use of lazy evaluation (reducing the number of CD calls by a factor of about 10 compared to Toggle PRM). In nearly all cases, the Lazy Toggle PRM methods outperformed their Lazy PRM counterparts due to the Toggle PRM techniques included in the Lazy Toggle PRM methods. Toggle uses fewer CD calls than Basic because Toggle uses effective strategies to solve the queries more intelligently, reducing the number of CD calls needed to map narrow passages.

Figure 4 illustrates the major downside to lazy methods — an increased number of shortest path graph searches (A^* in our experiments). Toggle PRM and Basic PRM always use one graph search per query; thus, they are not shown in the

Total Number of Successful Trials out of 30							
Environment	LazyToggleFree	LazyToggleMix	LazyToggle	Toggle	LazyFree	Lazy	Basic
S2D	30	30	30	30	30	26	30
Walls	29	29	28	30	30	2	30
2DMaze	30	30	25	30	30	4	30
MazeTunnel	30	30	30	30	30	2	30
ZTunnel	30	30	9	30	30	11	30

TABLE I: Successful trials for each method in the basic environments. Failures are due to excessive memory use.

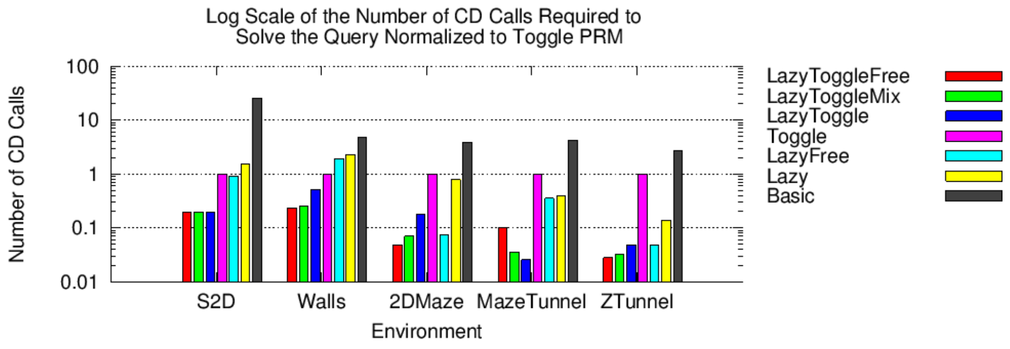


Fig. 3: Log scale of the total number of CD calls required to solve the query normalized to Toggle.

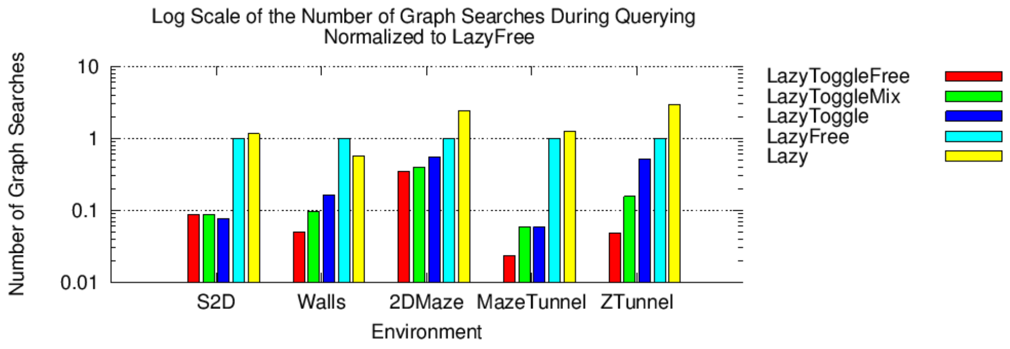


Fig. 4: Log scale of the total number of graph search operations needed to solve the query normalized to LazyFree. Toggle and Basic always use one graph search per query; thus, they are not shown.

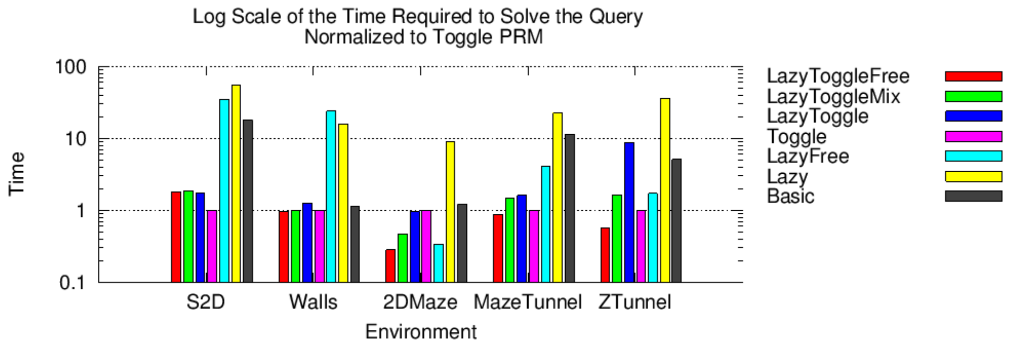


Fig. 5: Log scale of the total time needed to solve the query normalized to Toggle.

graph so that we may focus on the differences between the Lazy Toggle PRM and Lazy PRM methods. All lazy methods require more graph searches because they repeatedly extract paths from the graph. However, the Lazy Toggle PRM methods use much fewer graph searches than the Lazy PRM methods since they generally solve queries faster.

Finally, Figure 5 shows the overall efficiency of the methods tested. In S2D and Walls, two very simple environments (i.e., described by few polygons), Toggle PRM outperformed most or all of the Lazy Toggle PRM methods, and Basic outperformed the Lazy PRM methods. This is due to the increased number of graph searches needed to solve the problem with the lazy methods, even though the number of CD calls were significantly reduced. The trade-off then becomes a question of comparing the expense of graph searches to CD calls. If the number of polygons in the environment is low, making CD calls inexpensive (as in S2D and Walls), then the lazy strategy might not be an appropriate choice. On the other hand, when the CD calls outweigh graph searches, lazy methods show their strength. This is the case for the rest of the environments, where Lazy Toggle PRM generally outperforms the others in terms of total computation time (especially when the parameters are well-tuned, such as LazyToggleFree in MazeTunnel and ZTunnel). However, even in environments not particularly suited for lazy evaluation, namely those very basic environments, Lazy Toggle PRM is still among the best of the methods tested.

C. Complex Scenarios

As was seen in the previous section, lazy methods perform well when the complexity of the environment increases to the point where CD calls become a severe bottleneck in mapping an environment (compared to graph operations). Here, we compare the effectiveness of the lazy methods in two realistic and complex single-query examples (Helicopter and Building), each with a complex environment description. Results are averaged over successful runs. The number of successful trials out of 60 is shown in Table II, a log scale of the number of CD calls normalized to Toggle is shown in Figure 6, a log scale of the number of graph search calls normalized to LazyFree is shown in Figure 7, and a log scale of the total time required to solve the query normalized to Toggle is shown in Figure 8.

As demonstrated in the basic scenarios, Figure 6 shows that in the complex scenarios, lazy evaluation again reduced the number of CD calls: in each environment, the Lazy Toggle PRM methods outperformed Toggle and the Lazy PRM methods outperformed Basic (when excessive memory was not used). Also, as shown in Figure 7, lazy evaluation increases the number of graph searches used. As before, the Lazy Toggle PRM methods outperformed the Lazy PRM methods in terms of both CD calls and graph searches.

Figure 8 shows that ultimately Lazy Toggle PRM methods have a reduced computation time for complex scenarios compared to all other methods tested. Statistical analysis (omitted here for space) shows that each Lazy Toggle PRM method performs faster than the other methods with a significance level of $\alpha = 0.01$, with the average runtime two to four times as fast as the next best method in each environment. Note that the time required is directly related to the number and expense of CD calls and graph searches; thus, Lazy Toggle PRM improves the overall efficiency in these environments. In the Building environment, the existence of vastly different paths likely caused inefficiencies in the Lazy PRM methods since they might attempt to validate larger portions of the roadmap, but the Lazy Toggle PRM methods were not hindered. Also, the trend of increasing efficiency improvements with increasing environment complexity is promising since we expect applications to increase in complexity in the future.

V. CONCLUSION

In this paper, we present a novel combination of Lazy PRM and Toggle PRM, each method with its own benefits. The new algorithm, Lazy Toggle PRM, uses lazy evaluation in constructing a roadmap and the strengths of Toggle PRM to augment the roadmap. This enhancement technique is especially appropriate for narrow passages. We experimentally analyzed Lazy Toggle PRM in a wide array of difficult narrow passage environments, showing improved efficiency compared to Toggle PRM and Lazy PRM.

In the future, Lazy Toggle PRM could be expanded to use fuzzy evaluation, where probabilities of collisions are stored in each map. Validating paths with high probability of success will probably improve the efficiency of Lazy Toggle PRM. Additionally, exploring the use of Lazy Toggle PRM in changing and dynamic environments could prove successful.

REFERENCES

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Robotics: The Algorithmic Perspective*, pages 155–168, Natick, MA, 1998. A.K. Peters. Proc. Third Workshop on Algorithmic Foundations of Robotics (WAFR), Houston, TX, 1998.

[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] R. Bohlin and L. E. Kavraki. Path planning using Lazy PRM. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 521–528, 2000.

[4] 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.

[5] B. Burns and O. Brock. Sampling-based motion planning using predictive models. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3313–3318, 2005.

[6] B. Burns and O. Brock. Toward optimal configuration space sampling. In *Proc. Robotics: Sci. Sys. (RSS)*, pages 105–112, 2005.

[7] J. Denny and N. M. Amato. Toggle PRM: Simultaneous mapping of C-free and C-obstacle - a study in 2D -. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 2632–2639, San Francisco, California, USA, Septempber 2011.

[8] J. Denny and N. M. Amato. The toggle local planner for sampling-based motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1779–1786, St. Paul, Minnesota, USA, May 2012.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 473–479, 1999.

[14] 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.

[15] J.-M. Lien, S. L. Thomas, and N. M. Amato. A general framework for sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4439–4444, September 2003.

[16] 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.

Total Number of Successful Trials out of 60							
Environment	LazyToggleFree	LazyToggleMix	LazyToggle	Toggle	LazyFree	Lazy	Basic
Helicopter	60	58	58	60	60	31	60
Building	55	52	49	58	33	3	50

TABLE II: Successful trials for each method in the complex environments. Failures are due to excessive memory use.

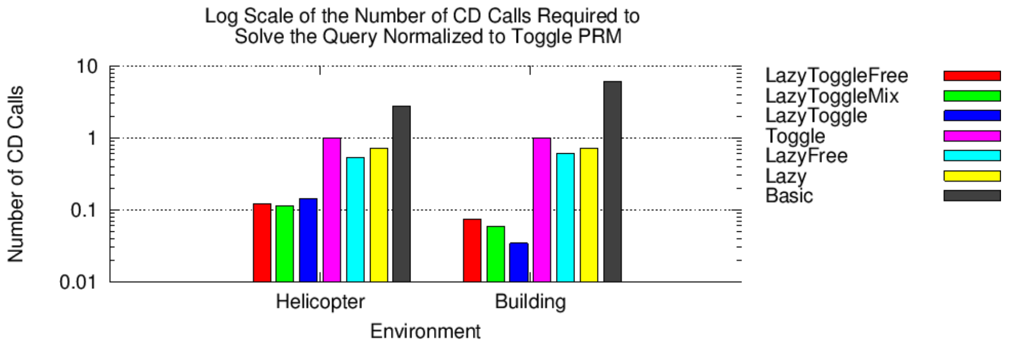


Fig. 6: Log scale of the total number of CD calls required to solve the query normalized to Toggle.

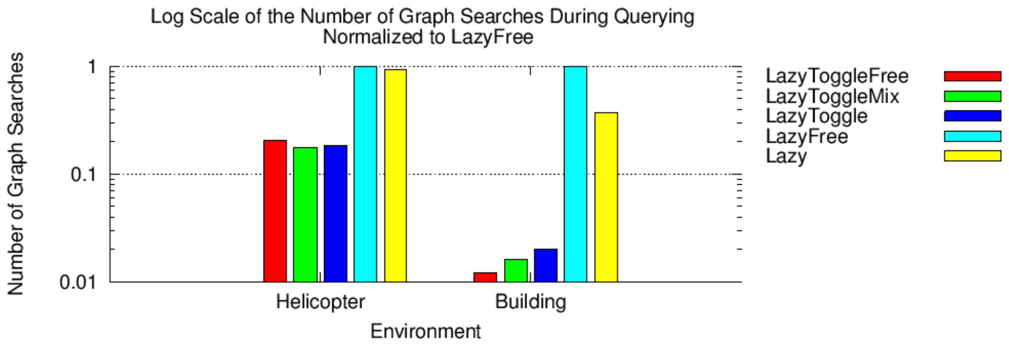


Fig. 7: Log scale of the total number of graph search operations needed to solve the query normalized to LazyFree. Toggle and Basic always use one graph search per query; thus, they are not shown.

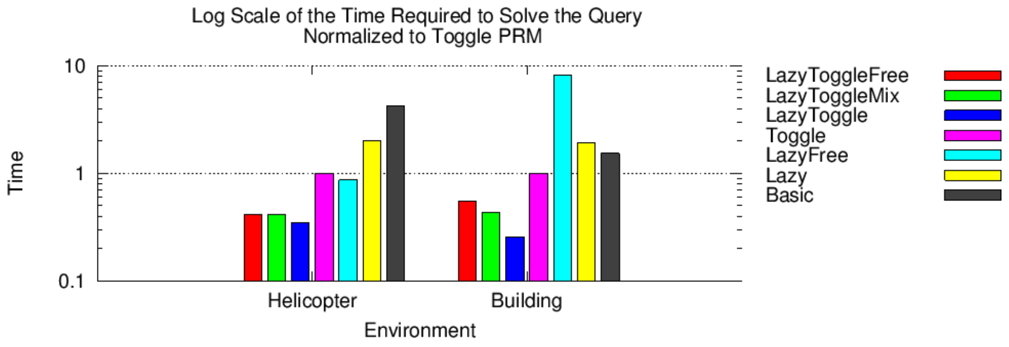


Fig. 8: Log scale of the total time needed to solve the query normalized to Toggle.

[17] C. L. Nielsen and L. E. Kavraki. A two level fuzzy PRM for manipulation planning. *IEEE/RSJ International Conference on Intelligent Robotics and Systems*, pages 1716–1722, 2000.

[18] 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.

[19] A. C. Shkolnik and R. Tedrake. Sample-based planning with volumes in configuration space. *CoRR*, abs/1109.3145, 2011.

[20] A. P. Singh, J.-C. Latombe, and D. L. Brutlag. A motion planning approach to flexible ligand binding. In *Int. Conf. on Intelligent Systems for Molecular Biology (ISMB)*, pages 252–261, 1999.

[21] G. Song, S. L. Miller, and N. M. Amato. Customizing PRM roadmaps at query time. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1500–1505, 2001.

[22] 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.

[23] 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.

[24] 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)*, Vilamoura, Algarve, Portugal, 2012.