An Empirical Study of Optimal Motion Planning

Jingru Luo and Kris Hauser

***Abstract*— This paper presents a systematic benchmarking comparison between optimal motion planners. Six planners representing the categories of sampling-based, grid-based, and trajectory optimization methods are compared on synthetic problems of varying dimensionality, number of homotopy classes, and width and length of narrow passages. Performance statistics are gathered on success and convergence rates, and performance variations with respect to geometric characteristics are analyzed. Based on this analysis, we recommend planners that are likely to perform well for certain problem classes, and draw recommendations for future planning research.**

I. Introduction

The *optimal motion planning* problem[1]asks to find a path of minimum cost (such as shortest length) that avoids constraints, and fast methods for solving them in high-dimensional, geometrically complex spaces would lead to major advances in robotics, structural biology, and computer animation. A flurry of recent work has studied three vastly different approximation approaches — sampling-based roadmap methods, numerical trajectory optimization, and grid-based search, producing dozens of planner variants, theoretical soundness and convergence proofs, and some evidence of practical success. But benchmarking data remains scarce, and few studies have compared different planners on equal footing. Hence the fundamental question "For a given problem (or class of problems), which planner is best?" is still wide open.

This paper aims to begin filling in these knowledge gaps by performing a systematic empirical study of optimal motion planning performance. To compare a wide variety of planners in our study, we address only the kinematically-constrained shortest-path problem. Six algorithms are compared: multiple-restart Randomly-exploring Random Tree (MRRT) [12], MRRT with shortcutting (MRRT+S), RRT*, Cross Entropy method (CE) [11], a resolution-complete variant of the Fast Marching Method (FMM), as well as a novel lazy variant of the PRM* algorithm (L-PRM*) [9]. These methods were chosen as representatives of sampling-based, optimization, and grid-based approaches, and can be run in "any-time" fashion with path cost improving asymptotically as more time is spent planning. Moreover, they accommodate varying dimension and "black box" constraint tests, which is a very general computational model.

It is widely known that planner performance depends strongly on configuration space (C-space) geometry. Certain

*This work is partially supported by NSF CAREER #1253553.

School of Informatics and Computing, Indiana University, Bloomington, IN 47405 USA {luojing,hauserk}@indiana.edu

[1]Essentially identical to optimal control, but the "motion planning" connotes a higher emphasis on complex constraints like obstacle avoidance.

TABLE I
SUMMARY OF PLANNERS AND RESULTS

Planner	Type	Reference	Results
FMM	Grid search	Adapted from [21]	Excellent in low-D spaces, consistent failure in $\geq 6D$
CE	Traj. opt.	[11]	Generally poor, somewhat competitive in high-D
MRRT	Sampling-based	[15]	Suboptimal, high variance
MRRT+S	Sampling-based & traj. opt.	[15]	Performance often excellent but uneven, good in 1 homotopy class
RRT*	Sampling-based	[9]	Good performance all around, slow convergence
L-PRM*	Sampling-based	Adapted from [9] and [3]	Good performance all around, least sensitive to CC time

effects are predicted by theory; e.g., grid-based planners run in time worst-case exponential in dimension, and the time to find a first solution in sampling-based planners is not directly tied to dimension but is instead highly dependent on narrow passage width [10], [12]. To study whether such behavior is found in practice, we devised several synthetic benchmark problems in which the following geometric characteristics can be carefully controlled:

1) Dimensionality,
2) Width of optimal narrow passage,
3) Length of optimal narrow passage,
4) Presence of "nuisance" suboptimal homotopy classes.

To our knowledge the effects of the last two characteristics have not yet been theoretically analyzed, but our study finds significant effects in practice.

Our study measures planner performance in terms of convergence plots, which capture the relation between computation time and the cost of the best path found by the planner. This allows a variety of time-dependent behaviors to be compared; for example, one planner may produce near-optimal solutions but with high initial cost, while another may quickly generate a suboptimal solution but additional computation fails to refine it much. Certain applications may prefer one behavior over another. We also measure fraction of time spent in feasibility testing versus overhead, and planners that spent less time in feasibility testing are more scalable to problems with complex tests.

From our results we draw several performance comparisons (Tab. I) and make high-level observations to suggest future directions for motion planning research. We observe that all techniques varied in their strengths, but the most consistent performer was MRRT+S. Because it combines sampling-based and trajectory optimization techniques, this result suggests that the development of other hybrid approaches may be a promising line of research for general-purpose planning. We also observe that most techniques work reliably in low-dimensional spaces with good visibility, but all techniques are unreliable at even moderately high-dimensional problems (5D+) with narrow passages. This suggests that future research in this field should focus on solving such "hard" problems.

[Code for all planners and benchmark problems studied in this paper will be made publicly available upon acceptance of this publication.]

II. Background

An early theoretical result on exact optimal motion planning proved its NP-hardness even in 3-D space with polygonal obstacles [20]. As a result, optimal motion planners typically focus on approximation techniques. They generally fall into three categories: sampling-based, trajectory optimization, and grid-based approaches.

Sampling-based optimal motion planners. Sampling-based motion planners operate by building a roadmap of simple paths that connect randomly-sampled feasible configurations, and then searching this network for a path connecting the start and goal. They have shown great success in finding feasible paths in high-dimensional, geometrically-complex problems [3], [10], [13], and are proven to be probabilistically complete: the planner will find a feasible path, if one exists, with probability 1 as more time is spent planning. However, they often produce jerky, high-cost paths and unpredictable results from one run to the next.

To address these weaknesses, researchers have recently turned to the problem of devising *asymptotically optimal planners* whose paths approach an optimum as more time is spent planning [9]. These planners continue to refine the roadmap after the first path is found in hope that a better path will be found. Several variants of RRT* have been presented in the literature, e.g., [1], [15], [18].

Grid-based approaches. Grid-based approaches convert the continuous space into a discrete grid which can be searched using heuristic search techniques. The ARA* method is an anytime variant of heuristic search that has been applied to mobile robots, and whose suboptimality (on the grid) approaches 0 as more time is spent planning [14]. However, a path on a grid constrained to $90°$ and $45°$ angles will in general not converge to the optimum in continuous space. Variants have been developed to generate the true optimal path on 2D grids [5], [17]. The Fast Marching Method (FMM) was introduced as a numerical method for solving the Eikonal equation [21], and has been applied to motion planning problems in 2D [7], [16]. It converges to an continuous-space path with suboptimality a linear function of grid resolution. To our knowledge, this paper is the first to consider any-time FMM and to apply FMM to path planning in more than 3 dimensions.

Trajectory optimization approaches. The third approach performs numerical optimization in the parameter space of a trajectory representation of fixed dimensionality, such as a list of intermediate points or the control points of a B-spline. These techniques start at an initial path, and choose directions in parameter space to descend the cost function (e.g. CHOMP [19]). They often converge quickly locally, but without a carefully designed initial path, easily fall prey to local minima. To escape local minima many methods inject stochasticity into the process, include simulated annealing [4], population-based approaches [2], the STOMP algorithm [8], and the Cross-Entropy method [11]. This study compares only the Cross-Entropy method because it is able to operate without the need to compute obstacle distances and penetration depths, which is in general a computationally complex operation.

III. Problem Statement

This paper considers time-invariant, geometrically-constrained, endpoint-constrained, shortest-path problems in bounded configuration spaces $\mathcal{C} = [0,1]^D$. Specifically, they ask to compute a path $y : [0,1] \rightarrow \mathcal{C}$ that solves:

$$\begin{gathered} \min_y Cost(y) \\ \text{such that} \\ y(0) = q_s, y(1) = q_g \\ y(u) \in \mathcal{F} \text{ for all } u \in [0,1]. \end{gathered} \tag{1}$$

where the cost function is path length, q_s and q_g are endpoints, and the *free space* $\mathcal{F}$ is the set of all configurations that satisfy geometric constraints. The *forbidden space* (or C-obstacle) is the complement of the free space.

Problem implementation. We supply planners with two problem-specific constraint testing subroutines:

- IsFeasible(q): tests whether q lies in free-space.
- IsVisible(q,q'): tests whether the straight-line path between q and q' lies in free-space.

All planners use a common implementation of these subroutines to ensure fair comparison.

Performance metric. Our experiments test optimization performance – length of the best path found — as a function of computation time limit and the geometric characteristics of benchmark problems. In other words, we estimate the distribution of the function *Best-Path-Length(planner,parameters,problem,time)*. For each benchmark problem, we selected a "middle point" of geometric characteristics as a base case, and then test performance changes with respect to individual problem characteristics.

Parameter tuning. Some algorithms (CE, MRRT, MRRT+S, RRT*) have parameters that must be tuned to achieve acceptable performance. To choose them, we tuned stable values for the base case scenario by ensuring that factor-of-two increases and decreases of parameters led to performance loss or had insignificant effects on performance. Stable choices were applied across each problem variant in the same class of benchmarks.

IV. Benchmark Problems

To study the effects of geometry on planning performance, we consider four benchmark problems with variable dimension and visibility properties.

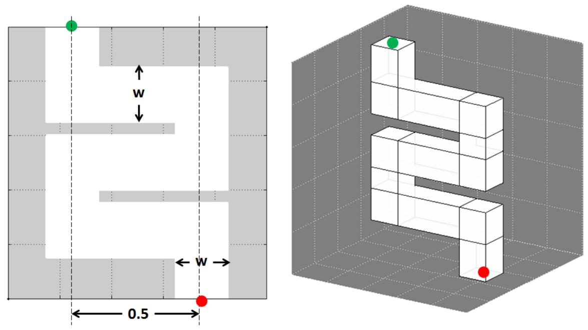

Fig. 1. Benchmark 1: a complex narrow passage with width W shown in $D = 2$ and $D = 3$. Red and green dot indicate start and goal configuration respectively.

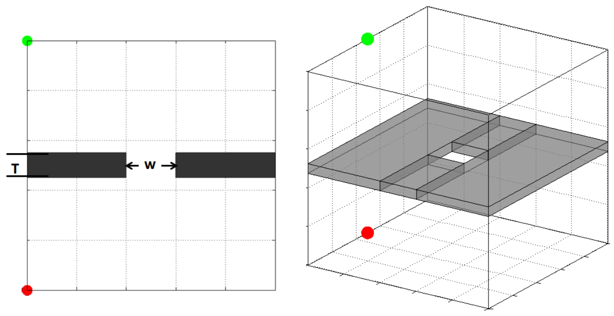

Fig. 2. Benchmark 2: a narrow passage with width W and thickness T shown in $D = 2$ and $D = 3$. Red and green dots indicate start and goal configurations.

A. Benchmark #1: Kink

Benchmark 1 is a tortuous narrow passage. A planner needs to turn at least 6 corners to find a path from start to goal (Fig.1). The start and goal are fixed to $(0.75, 0.5, ..., 0.5, 0)$ and $(0.25, 0.5, ..., 0.5, 1)$, and the width of the passage is defined by a parameter W. We tested characteristics $D \in \{2, \ldots, 5\}$ and width $W \in [0.05, 0.2]$, with $D = 3$ and $W = 0.15$ chosen as the base case.

B. Benchmark #2: One-Class Hole

Benchmark 2 is a narrow passage problem with non-uniform visibility properties and a single homotopy class (Fig. 2). The start and goal are fixed to $(0, 0.5, 0, 5, ..., 0, 5, 0)$ and $(0, 0.5, 0, 5, ..., 0, 5, 1)$ respectively. The narrow passage has width W on $n - 1$ axes and and length T on the remaining axis. The theoretical best path can be calculated given width W and thickness T. We tested characteristics $D \in \{2, \ldots, 5\}$, $W \in [0.005, 0.1]$, and $T \in [0.01, 0.5]$, with $D = 3$, $W = 0.05$ and $T = 0.1$ chosen as the base case.

C. Benchmark #3: Two-Class Hole

Benchmark 3 is identical to Benchmark 2 except that it introduces a wide, suboptimal homotopy class (Fig. 3). This "nuisance" homotopy class may lead the planner to waste time optimizing useless paths, so this example helps study planners' ability to find hard global optima vs. easy local optima. The optimal homotopy class passes through a narrow passage with width W on $n - 1$ axes and length T on the remaining axis. The suboptimal homotopy class is a much wider passage, width 0.5 on one axis and width 1 on the remaining $n-2$ axes. We tested the same characteristics and base case as in Benchmark 2.

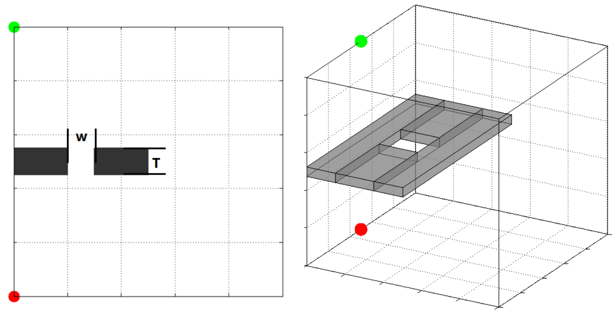

Fig. 3. Benchmark 3: two-class narrow passages, with one controlled by the width W and obstacle thickness T, and a wider constant passage. Shown in $D = 2$ and $D = 3$.

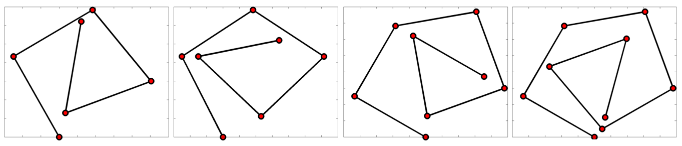

Fig. 4. Benchmark 4: planar linkages with 5 – 8 joints must be unfolded.

D. Benchmark #4: Planar Linkage

Benchmark 4 is a planar linkage with D joints folded. The goal is to unfold the linkage so that all the links become horizontally extended. The starting configurations are displayed in the Fig. 4 and the goal is $(\pi, 0, ..., 0)$. The configuration space contains a single homotopy class and a narrow passage as the first few joints needs to be unfolded first in order to avoid collision. We tested $D \in \{5, 6, 7, 8\}$.

E. Collision Checking

In benchmarks 1–3, C-obstacles are formulated as polytopes in half-space representation, and in benchmark 4, C-obstacles are detected by pairwise segment collision checking. The validity of a path is checked by discretizing the path using repeated bisection and checking the validity of intermediate states until a resolution δq is reached. In this work, all the algorithms use the same resolution $\delta q = 0.001$.

V. Overview of Planning Algorithms

Here we describe the planners presented in Tab. I. All algorithms are compared on identical problem implementations given the IsFeasible and IsVisible subroutines. Implementations of RRT, RRT*, RRT+S are taken from OMPL (Open Motion Planning Library) [22] and we integrated custom implementations of CE, FMM, and L-PRM* into OMPL for comparison.

A. Fast Marching Method

The Fast Marching Method (FMM) is a grid-based technique that approximates the true shortest paths in continuous space [21]. It proceeds like a standard grid search but uses a

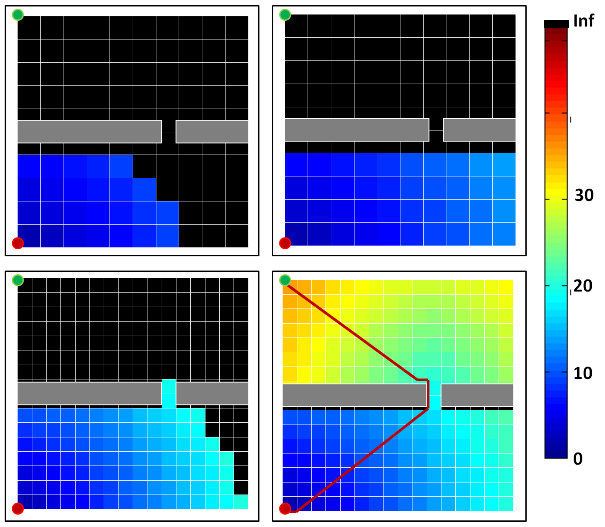

Fig. 5. Illustrating FMM. A distance map $D(q)$ is computed on the grid, initializing the start node to $D(q_s) = 0$ while $D(q) = \infty$ for all other nodes. It then propagates distances outward from the search fringe, and if goal is reached, a solution path can be computed by the following the gradient. This process is embedded in an outer loop that progressively increases the grid resolution and maintains the shortest feasible path.

special cost propagation step based on simplex interpolation. We use a resolution-complete version of FMM that performs multiple searches with progressively finer grids. Although FMM is typically used in 2 or 3 dimensions, we present an any-dimension implementation that is not well known in the robotics community.

First we describe the fixed-resolution algorithm (see Fig. 5). A grid of resolution h is defined over the space. Each grid point q is given a movement cost $C(q)$ equal to 1 if q is collision free, and ∞ otherwise. FMM computes a distance map $D(q)$ on the grid by propagating distances outward from the start grid point to neighbors in a brush-fire fashion until the goal is reached. FMM maintains a status $S(q)$ of each point, which can be one of three categories:

1) Alive: inner nodes of the brush fire, for which $D(q)$ has been determined,
2) Close: nodes at the brush-fire fringe, where $D(q)$ is still being determined via propagation from adjacent Alive and Close nodes,
3) Far: outer nodes, not adjacent to Alive nodes.

The set of Close points is stored in in a priority queue (min-heap) sorted by value $D(q)$. The algorithm proceeds by extracting the nearest Close point, marking it as Alive, and computing new candidate distance values are computed for its Close and Far neighbors. Those neighbors are marked as Close and added to the queue. This repeats until the goal is reached, at which point the **Descent**() routine extracts an approximation of the shortest path by performing gradient descent of $D(q)$.

The algorithm is summarized below:

Algorithm 1 Fast Marching Method

```
1:  Initialize $D(q) \leftarrow \infty$ for all $q$.
2:  Initialize Alive $\leftarrow \{\}$
3:  Initialize Close $\leftarrow \{q_s\}$, $D(q_s) \leftarrow 0$;
4:  while Close is not empty do
5:      $q \leftarrow$ node in Close with minimum $D(q)$
6:      Add $q$ to Alive
7:      if $q$ = $q_g$ then return Descend()
8:      for each neighbor $p$ of $q$ that are not Alive do
9:          if $p$ is in Far then
10:             $D(p) \leftarrow D(q) + C(p)$, add $p$ to Close
11:         if $p$ is in Close then
12:             Set $D(p) \leftarrow \min(D(p)$,minDistance$(q,p))$
```

The function **minDistance**(q,p) computes a distance value $D(p)$ as propagated from distance values from q and other neighboring Alive grid points. Rather than simply adding length along a grid axis, it approximates the true optimal cost from the start. Consider a grid cell c adjacent to both p and q, and let $q_1, \ldots, q_m$ be a subset of alive vertices of this cell. These vertices form a simplex s (either a point, or line segment in 2D, and a point, segment, or triangle in 3D). Let $d_1, ..., d_m$ be the distances values at the simplex vertices. The key assumption is that across the simplex, the distance function is given by a barycentric interpolation across $d_1, \ldots, d_m$. If $u = (u_1, ..., u_m)$ are the barycentric coordinates, the interpolated value is simply $d(u) = u_1 d_1 + \cdots + u_m d_m$. Hence the minimum distance at p through this simplex is equal to minimum of $d(u) + C(u)\|u\|$ for all barycentric coordinates.

To find this value we solve the optimization problem:

$$\min_u f(u) = u_1 d_1 + \cdots + u_m d_m + C(n)\|u\| \quad (2)$$

$$\sum_{i=1}^{m} u_i = 1 \quad (3)$$

Lagrange multipliers are used to solve for the critical points of the minimization problem analytically. Note that this must be performed for every simplex of Alive vertices containing q; the minimum value is reported for the final output of minDistance(q,p).

The parameter h plays an important role, as finer grids yield better paths but increase computational cost. Hence, we run a resolution-complete version of FMM in which fixed-resolution FMM is run multiple times, with each iteration reducing h by a factor of $2^{1/D}$. This ensures that the number of grid cells roughly doubles upon each iteration.

As presented, FMM fails to detect obstacles between grid points. Hence, we check the validity of the path extracted by Descend(). If collision checking fails, we remember the infeasible configuration by adding it to a list. In each subsequent FMM iteration, for each configuration q_{infeas} in that set, we turn the grid cell containing q_{infeas} into a small obstacle by marking its vertices with infinite cost.

B. Cross-Entropy Randomized Motion Planning

Cross-Entropy randomized motion planning (CE) is a trajectory optimization approach which estimates a probability

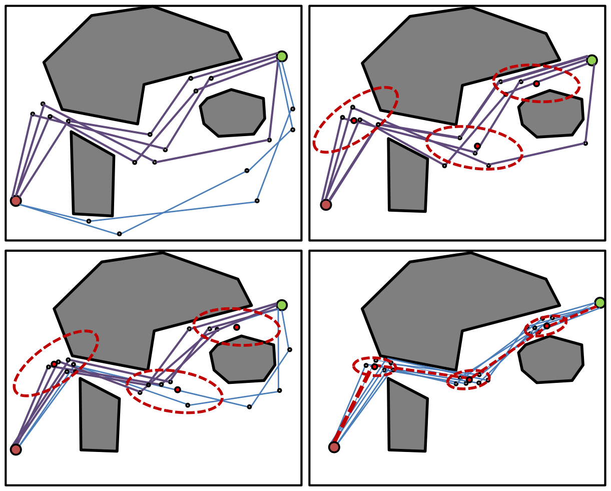

Fig. 6. Illustrating CE. Paths are parameterized with 3 intermediate states between start and goal, and a estimated distribution of optimal paths is modeled a multivariate Gaussian distribution. Given a sample of N feasible paths from sampled from that distribution, the best ρ fraction of paths are retained to estimate the optimal path distribution on the next iteration. This repeats until the estimate converges.

distribution over optimal paths, and iteratively improves this estimation until it converges. It is initialized with a set of N sampled or provided initial paths, and the probability distribution of optimal paths is estimated from the best ρ fraction of this set (the so called *elite set*). This distribution is used to sample a new set of N feasible paths, and this process repeats until convergence is reached. The rationale behind this approach is that subsequent distributions become more accurate representations of the optimal path, and hence paths sampled from this distribution are guided towards the optimum. Fig. 6 illustrates this process.

Our implementation followed the description in [11]. Trajectories are parameterized as a sequence of n intermediate milestones between the start and goal, connected in piecewise linear fashion. Multivariate Gaussians were used as the probability distribution model. The sample of initial paths was generated via random sampling. We report the feasible path sample with lowest cost.

The set of algorithm parameters is listed as follows:

1) n: number of intermediate states along the trajectory. We set $n = 6$ for Benchmark #1 as the path needs to turn at least 6 corners, and $n = 3$ from experimental tuning for Benchmark #2 and #3.
2) N: number of sample paths. A high value requires a large number of paths to be sampled initially while a low value does not give enough information to learn the probability distribution. We set it to $(n \cdot D)/\rho$.
3) ρ: the fraction of trajectories retained in each elite set. From tuning, $\rho = 0.2$ is used for Benchmark #2, $\rho = 0.05$ for #3, and $\rho = 0.5$ for #1 and #4.

C. RRT with Restarts and Shortcutting

The Rapidly-exploring Random Tree (RRT) planner explores the free space by incrementally growing a tree rooted at the start configuration [12]. In each iteration, it draws a free random sample and steers the nearest tree node towards the random sample (see Fig. 7). With some probability, an extension is drawn from the closest node to the goal. The original version of RRT searches for a feasible path, not an optimal. Multiple-restart RRT converts it into an optimizer by running RRT multiple times and record the best solution [15].

RRT is requires tuning the following parameters:

1) S: steering length, the maximum distance that a tree node is extended towards the target point.
2) G: goal bias, probability of choosing the goal as the target point rather than a random point.

Our tuning found stable values at $S = 0.2\sqrt{D}$ and $G = 0.05$ for the tested problems.

MRRT finds feasible solutions quickly but generates jerky paths, so an intuitive idea is to combine it with a small amount of optimization. In the MRRT+S case, this takes the form of "shortcutting" [15]: every time a solution is found by RRT, MRRT+S spends time optimizing the best solution found so far by removing unnecessary path segments. Each shortcutting step repeatedly samples a random section of the path and connects its end points with a straight line if the segment is collision free [15]. Every time a solution path is found by RRT, we shortcut the best solution found so far (Alg. 2). There is a tradeoff in performing more vs less shortcutting; our implementation attempts the same number of shortcuts as there are vertices in the path. Performance was relatively insensitive to changes in this parameter.

Algorithm 2 MRRT+S

```
1: $P_{best} \leftarrow \varnothing$
2: for i=1,2,... do
3:     $P_i \leftarrow$ RRT($q_s$, $q_g$)
4:     if $P_{best} = \varnothing \vee$ Cost($P_i$) $<$ Cost($P_{best}$) then
5:         $P_{best} \leftarrow P_i$
6:     $P_{best} \leftarrow$ Shortcut($P_{best}$)
```

D. RRT*

RRT* is an extension of RRT that ensures asymptotic optimality [9] by continuing to refine the tree by adding new milestones even after the first solution is found. A key contribution is a "rewiring" step that maintains approximate shortest paths to every milestone in the tree. To do so, it considers a neighborhood of each newly added milestone, and sets the parent to the one that would lead to the minimum cost from the root. Moreover, edges leading from the new milestone may lead to improved solution costs at the neighbors, so connections and rewiring steps are attempted at all neighbors and propagated through the tree (Fig. 7).

A key theoretical contribution is that in order for the solution to asymptotically approach the optimum, this neighborhood must have radius at least $R = \gamma \cdot (\log(|V|)/|V|)^{1/D}$ where $|V|$ is the number of nodes in the tree and γ is set to the maximum extent of the space, e.g., $\sqrt{D}$ in our hypercube. RRT* also contains the same parameters as RRT.

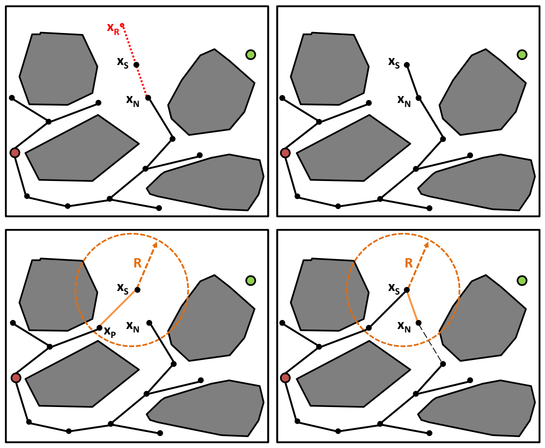

Fig. 7. Illustrating RRT and RRT*. Top row illustrates RRT: a random point x_R is sampled and its nearest neighbor x_N in the tree is extended toward x_R by a predefined steering distance S to obtain x_S. Bottom row illustrates RRT*: after obtaining x_S, it searches a neighborhood of radius R around x_S to find the a parent x_P that leads to the lowest cost from the root. The rewiring step updates best distances to nodes in the neighborhood that pass through x_S.

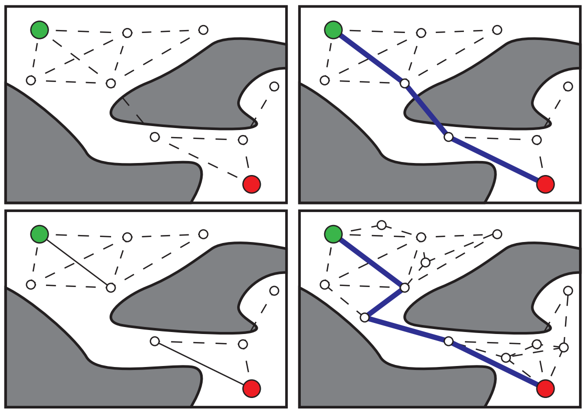

Fig. 8. Illustrating Lazy-PRM*. A PRM* roadmap is built between feasible milestones, but edges are not checked for collision until it finds a candidate path to the goal with lower cost than the previous best feasible path. The candidate path is checked for collisions, and any infeasible edges are removed from the graph. If feasible, the path is stored. The process repeats until convergence. This often leads to fewer edge collision checks than standard PRM*.

E. Lazy-PRM*

PRM*, also presented by [9], is a variant of the standard Probabilistic Roadmap technique that uses the same variable neighborhood trick as RRT*, and is also proven to converge asymptotically to an optimal path. We present Lazy-PRM*, a novel variant of PRM* that integrates a "lazy" technique from the Lazy-PRM algorithm [3]: collision checking of paths (via the IsVisible subroutine) is delayed unless they are part of a candidate best path to the goal (Fig. 8). This is potentially helpful because edge collision checking is one of the more expensive subroutines in a planner, and most edges are irrelevant to improving the optimal path. However, failing to perform edge checks early on may lead the planner to unnecessarily check paths whose endpoints straddle obstacles.

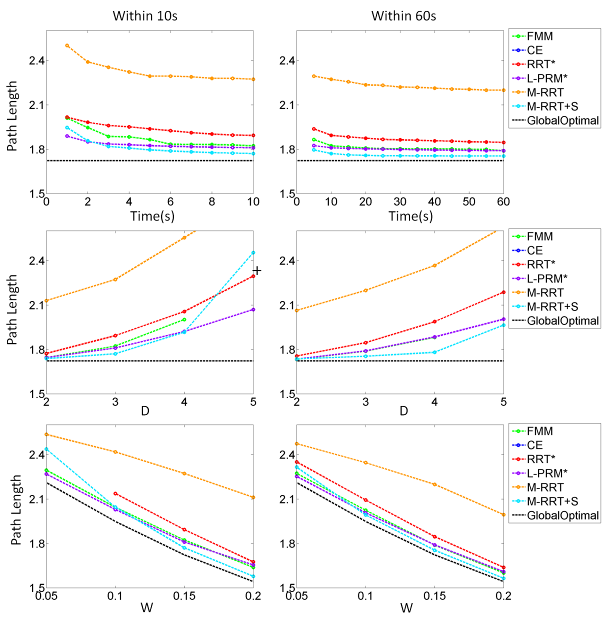

Fig. 9. Results on Benchmark #1. CE failed on all instances. + indicates lower than 50% success rate. Curves for the FMM and L-PRM* planners on the D plot are overlapping.

Every time a new path in the roadmap improves upon the best previously found path, all the edges on the shortest path leading from start to goal (that have not already been checked) are checked for collision. Infeasible edges are then removed from the roadmap and the algorithm continues. It is important to efficiently update shortest paths upon addition and deletion of edges in the graph, and we use dynamic shortest path algorithms for doing so [6].

VI. Results

A. Methodology

All experiments are performed on identical individual cores of an Odin cluster at Indiana University running Linux, each with 1.0GHz AMD Opteron processor and 4 GB memory. Each trial was performed with a time limit of 60 seconds. This limit was chosen relatively low because most robotics applications require results relatively quickly (seconds or minutes rather than hours), and our benchmarks are relatively simple compared to realistic problems. For each algorithm and benchmark, 20 trials were run to compute the mean and variance of path cost. Although FMM is deterministic, we perturbed the grid origin at random on each trial, which avoids artifacts in which the grid is luckily aligned with a narrow passage. Figures report results at short (10 s) and long (60 s) time cutoffs.

B. Convergence Plots

Benchmark 1 results are shown in Fig. 9. Here, FMM, MRRT+S RRT*, and L-PRM* perform well, with MRRT+S generally beating FMM and L-PRM* by a small margin. However, with very narrow passages it performs worse.

Benchmark 2 results are shown in Fig. 10. Here, FMM and MRRT+S are the most reliable planners. MRRT+S is

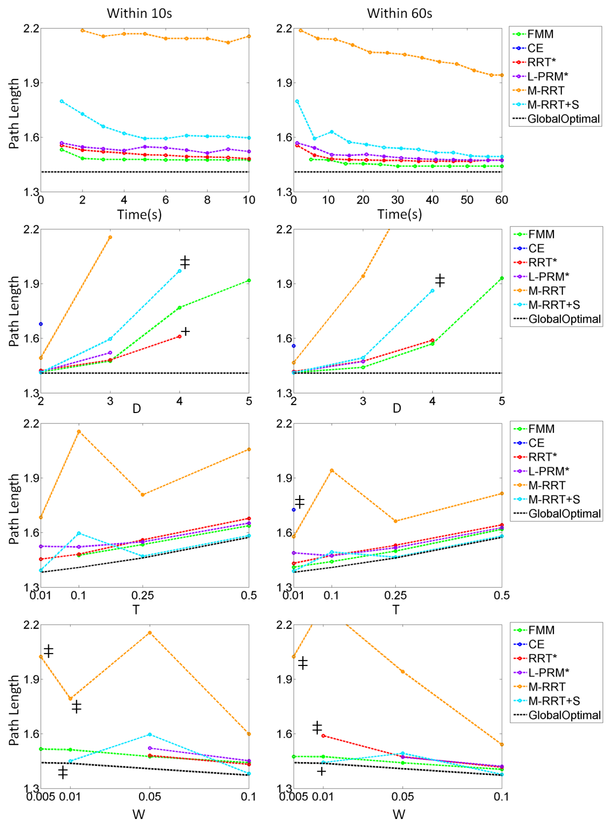
Fig. 10. Results on Benchmark #2. CE failed on almost all instances. + indicates lower than 50% success rate; ‡ indicates lower than 25% success rate.

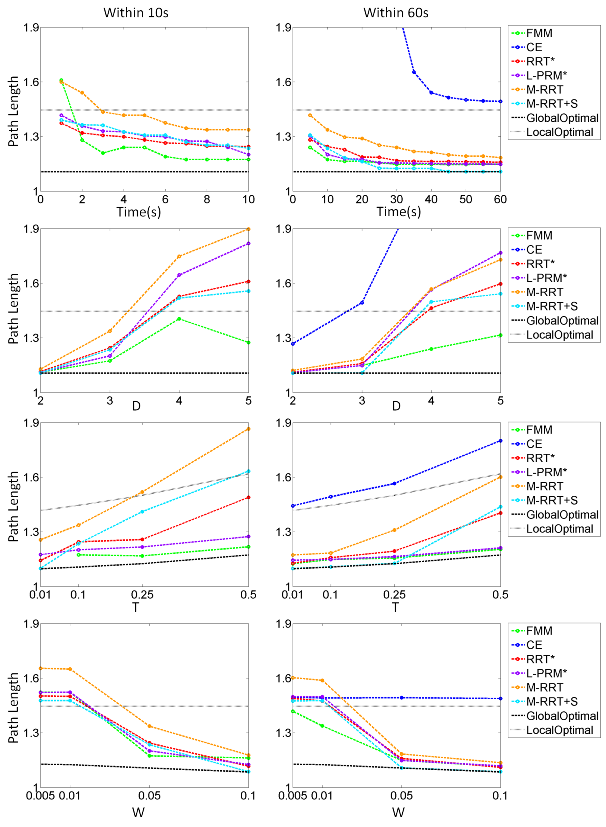
Fig. 11. Results on Benchmark #3. CE failed on all instances with 10 s cutoff.

often the best, but its performance has higher variance than FMM. Another interesting trend is that L-PRM* is thickness-sensitive, with worse performance as obstacles grow thinner. This is explained by the fact that lazy approaches often miss thin obstacles until the path checking stages, causing repeated calls for shortest path updates and path checking.

Benchmark 3 results are shown in Fig. 11. Here, FMM is a consistent high performer, and is able to discover the optimal homotopy class except for higher dimensions and the narrowest passages. Again, MRRT+S occasionally converges quickly to local optima, but finds the global optimum less frequently due to the presence of the nuisance homotopy class.

Another interesting finding is that MRRT, MRRT+S, and RRT* are thickness-sensitive, with thicker obstacles causing more diversion of effort into the nuisance homotopy class. This can be explained by the fact that extending a tree through narrow passages require a *sequence* of several luckily sampled configurations, which drops in probability as the passage grows longer. FMM is actually negatively thickness-sensitive, facing a similar problem as L-PRM* in Benchmark 2.

Benchmark 4 results are shown in Fig. 12. Here, the results are surprisingly different from before: FMM does drastically worse, rarely finding a solution, and CE becomes competitive with RRT* after some time. RRT* tends fails to refine the first path found, and bizarrely performs better as dimensionality grows, probably because it is approaching the pure exploration behavior of RRT. MRRT and MRRT+S are consistent winners.

C. Discussion

We observe a few trends from our study, and remark on some limitations of this work.

Planner strengths and weaknesses. Grid search works well in low-dimensional problems, but performance drastically degrades as dimension grows. Trajectory optimization is highly susceptible to "nuisance" homotopy classes. Sampling-based planners are fairly consistent across dimensions. However, performance advantages vary greatly as problem characteristics change — sometimes by orders of magnitude. The hybrid approach, MRRT+S, was most consistent. This suggests that hybrid techniques or problem adaptation may be a fruitful direction for future work.

Passage length and homotopy class sensitivity. To our knowledge, performance effects of narrow passage length and nuisance homotopy classes have not been explored in theoretical results. These factors should be investigated.

Collision checking time. We also find that the lazy planners L-PRM* and FMM spend a much smaller fraction of time collision checking (1.5% and 13%, respectively), because they delay edge visibility checking. Hence, their performance is less sensitive to constraint checking cost com-

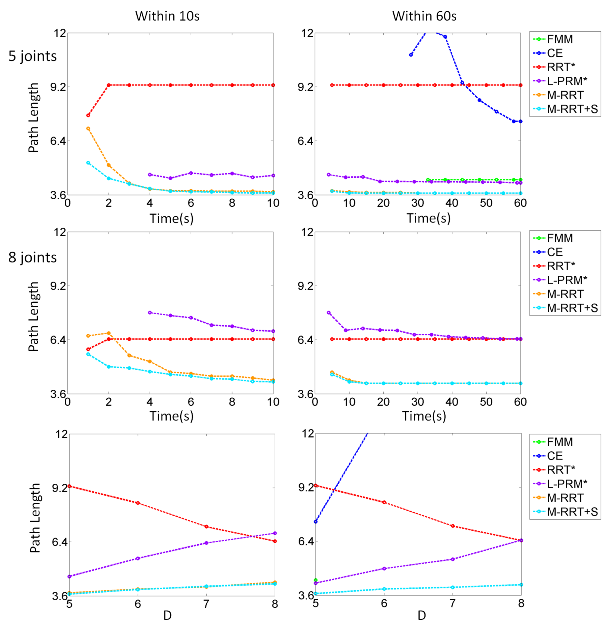

Fig. 12. Results on Benchmark #4. FMM and CE failed on all instances with 10 s cutoff, and FMM failed for $D \geq 6$. Curves for MRRT and MRRT+S are overlapping in the lower left plot. The path length in RRT* appears to grow because some trials found first paths later than others, and these paths were of worse quality.

pared to other methods. This is significant because collision checks are more time-consuming in more realistic problems, and hence their performance will not drop as strongly as the other methods, which spend upward of 40% of running time performing collision checking.

Extensions to other optimal planning problems. The scope of our study is limited to shortest-path, geometrically-constrained problems. Certain planners, such as MRRT and CE, easily handle other cost functions and differential constraints, while others, such as MRRT+S, RRT*, and grid-based methods require non-trivial modifications to handle such problems.

Richer constraint representations. Our study is also limited to the *black box constraint model* where membership in the free-space can be tested in boolean fashion. In other words, a planner can test *whether* $q \in \mathcal{F}$ but not how far or deep it is from obstacles. This is the least restrictive computational model, and the one most widely used in motion planning. However, some planners that operate on alternate representations that permit obstacle distance / penetration depth tests (e.g., CHOMP [19]). It would be worthwhile to study how obstacle representation could provide more powerful sources of information to guide planning in hard problems. In fact, it appears necessary to do so in order to tackle the challenge of high-dimensional spaces with narrow passages.

VII. Conclusion

This paper presented a systematic benchmark of optimal motion planners in the shortest-path, geometrically constrained setting. Planner performance was studied as a function of problem geometry as well as timing characteristics. Performance trends were analyzed, and recommendations were made for future directions in the field. In future work, we are interested in studying how these benchmarking comparisons correlate with complex real-world problems. We also intend to use this data to build "suggesters" for automatically choosing the best planner for a given problem.

References

[1] B. Akgun and M. Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *IEEE/RSJ Int. Conf. Intel. Rob. Sys.*, pages 2640–2645, 2011.

[2] J. T. Betts. Survey of numerical methods for trajectory optimization. *J. Guidance, Control, and Dynamics*, 21(2):193–207, 1998.

[3] R. Bohlin and L. Kavraki. Path planning using lazy prm. In *IEEE Int. Conf. Rob. Aut.*, volume 1, pages 521 –528 vol.1, 2000.

[4] W. Carriker, P. Khosla, and B. Krogh. The use of simulated annealing to solve the mobile manipulator path planning problem. In *IEEE Int. Conf. Rob. Aut.*, pages 204 –209 vol.1, may 1990.

[5] D. Ferguson and A. Stentz. Using interpolation to improve path planning: The field d* algorithm. *J. Field Rob.*, 23(2):79–101, 2006.

[6] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, 34(2):251 – 281, 2000.

[7] S. Garrido, L. Moreno, M. Abderrahim, and F. Martin. Path planning for mobile robot navigation using voronoi diagram and fast marching. In *IEEE/RSJ Int. Conf. Intel. Rob. Sys.*, pages 2376–2381, 2006.

[8] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE Int. Conf. Rob. Aut.*, pages 4569–4574, 2011.

[9] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7):846–894, 2011.

[10] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE T. Rob. Aut.*, 12(4):566 –580, Aug. 1996.

[11] M. Kobilarov. Cross-entropy randomized motion planning. In *Robotics: Science and Systems*, Los Angeles, USA, 2011.

[12] S. LaValle and J. Kuffner, J.J. Randomized kinodynamic planning. In *IEEE Int. Conf. Rob. Aut.*, volume 1, pages 473–479 vol.1, 1999.

[13] S. M. LaValle, J. J. Kuffner, and Jr. Rapidly-exploring random trees: Progress and prospects, 2000.

[14] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* search with provable bounds on sub-optimality. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Conf. Neural Information Processing Systems (NIPS)*. MIT Press, 2003.

[15] R. Luna, I. Sucan, M. Moll, and L. Kavraki. Anytime solution optimization for sampling-based motion planning. In *IEEE Int. Conf. Rob. Aut.*, pages 5068–5074, 2013.

[16] P. Melchior, B. Orsoni, O. Lavialle, A. Poty, and A. Oustaloup. Consideration of obstacle danger level in path planning using a* and fast-marching optimisation: comparative study. *Signal processing*, 83(11):2387–2396, 2003.

[17] A. Nash, K. Daniel, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. In *AAAI Conf. on Artificial Intelligence*, pages 1177–1183, 2007.

[18] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez. Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *IEEE Int. Conf. Rob. Aut.*, pages 2537–2542. IEEE, 2012.

[19] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE Int. Conf. Rob. Aut.*, pages 489–494. IEEE, 2009.

[20] J. H. Reif. Complexity of the mover's problem and generalizations. In *IEEE Conf. Foundations of Computer Science*, pages 421–427, 1979.

[21] J. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proc. National Academy of Science*, 1996.

[22] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. `http://ompl.kavrakilab.org`.