2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

Lazy Collision Checking in Asymptotically-Optimal Motion Planning

Kris Hauser

Abstract— **Asymptotically-optimal sampling-based motion planners, like RRT*, perform vast amounts of collision checking, and are hence rather slow to converge in complex problems where collision checking is relatively expensive. This paper presents two novel motion planners, Lazy-PRM* and Lazy-RRG*, that eliminate the majority of collision checks using a lazy strategy. They are sampling-based, any-time, and asymptotically complete algorithms that grow a network of feasible vertices connected by edges. Edges are not immediately checked for collision, but rather are checked only when a better path to the goal is found. This strategy avoids checking the vast majority of edges that have no chance of being on an optimal path. Experiments show that the new methods converge toward the optimum substantially faster than existing planners on rigid body path planning and robot manipulation problems.**

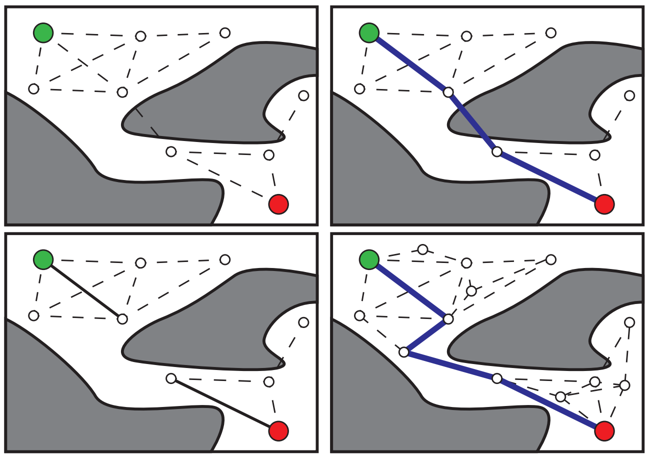

Fig. 1. Illustrating the lazy planner. Edges in the roadmap are not checked for collision (dotted lines) until a candidate path to goal is found. Feasible edges are marked (solid lines) and infeasible ones are deleted. The process repeats until a feasible path to the goal is found.

I. Introduction

The basic optimal motion planning problem asks to find a cost-minimizing path subject to obstacle avoidance constraints, and has been a highly active research topic since the development of sampling-based planners with proven asymptotic optimality properties (e.g., RRT* [11]). It has been known for a long time that sampling-based planners work well in high-dimensional spaces for the *feasible* motion planning problem, because their running time is not explicitly dependent on dimensionality [9]. It could be argued, then, that they should also work well for the *optimal* motion planning problem in many dimensions. But practical experience has demonstrated rather slow convergence to the optimum, especially as dimension increases [14].

Convergence is typically slow because, once a path is found, sampling-based planners need to sample a huge number of configurations before finding one that shortens the current best path. Each additional sample adds to the cumulative cost of collision checking, because testing is done on both the configuration and edges leading to nearby configurations. This is wasteful, since most work is being performed on elements with no opportunity to be part of an optimal path. Specifically, as the path approaches optimality, the region of free space that might contain a shorter one approaches a set of measure zero.

Inspired by this observation, this paper proposes a lazy strategy [4] to drastically reduce the fraction of edges that undergo collision checking. This benefit is substantial because edge collision checking accounts for the majority of collision checking time. It presents two planners, Lazy-PRM* and Lazy-RRG*, which, like their namesakes PRM* and RRG*, build a roadmap of feasible configurations (known as milestones), but with the difference that edges are not immediately checked for collision. If a better candidate path to the goal is found, then all edges along that path are checked for collision. Edges that fail the check are removed from the roadmap (Fig. 1). This strategy eliminates most of the cost of collision checking, at the cost of greater bookkeeping of shortest paths along the roadmap. We use a dynamic shortest paths algorithm [6] and other strategies to reduce the overhead of recomputing shortest paths.

Lazy-PRM* and Lazy-RRG* are any-time and asymptotically-optimal. But perhaps the most important property is that the resulting algorithm is relatively insensitive to the computational cost of collision checking subroutines, which is especially important in real-world, complex environments. Experiments on rigid-body and robot manipulator planning problems up to 12D suggest that Lazy-PRM* and Lazy-RRG* converges up to several orders of magnitude faster than PRM*, RRT*, and other asymptotically-optimal planners. Code for both algorithms is available in the Klamp't toolkit at `http://klampt.org`.

II. Related Work

Sampling-based motion planners build a roadmap of configurations, called milestones, sampled from the free space connected by straight line paths [12], [13]. Paths on the roadmap connecting the start and goal correspond to feasible solution paths in the free space. Originally applied to the feasible motion planning problem, in which the cost of the path is ignored, researchers have more recently applied

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

Pratt School of Engineering, Duke University, Durham, NC 27708 USA
`kris.hauser@duke.edu`

these methods to the optimal planning problem, in which a minimum cost path is desired (e.g., the shortest path).

In the *feasible* regime, sampling-based planners enjoy theoretical guarantees of probabilistically completeness, in that the probability of finding a feasible path, if one exists, approaches 1 as more milestones are sampled [12], [13]. Moreover they have state-of-the-art empirical performance in finding feasible paths in high-dimensional spaces. Unlike methods based on grid search or cell decomposition, which run in time exponential in dimension, the expected time to find a feasible path is not directly dependent on dimensionality, but rather on the visibility characteristics of the free space [9].

As a result, there has been enormous interest in applying sampling-based methods to the *optimal* motion planning problem. Karaman and Frazzoli recently presented the RRT* tree-growing algorithm and the RRG* and PRM* graph-growing algorithms, which are proven to be asymptotically optimal [11]. Their key contribution uses a result from the theory of percolation in random graphs to determine connection neighborhoods that guarantee that the shortest path in the roadmap approaches the optimal path in free space. However, theory predicts $O(1/\epsilon^d)$ samples are needed to achieve an ϵ-suboptimal path in d dimensions, which is no better than grid search, and experiments confirm that convergence rate is rather slow in high dimensional spaces.

As a result, recent work aims to improve convergence rate. Early convergence can be accelerated by checking fewer edges to arrive at suboptimal solutions faster [2], with recent approaches provably converging toward near-optimality (e.g., LBT-RRT* [15], SPARS [5]). Other approaches reduce time-per-step once the roadmap grows dense, either by using sampling strategies that place samples where they are more likely to yield an improvement in the optimal path [7], or by using C-obstacle distance queries to incrementally cover free-space with balls that are known to be collision free, which allows collision checks to be skipped for samples near existing milestones [3].

Lazy edge collision checking has been used in the past to speed up sampling-based motion planners, most notably Lazy-PRM [4] and SBL [16]. It is effective because edge checking is typically one or two orders of magnitude more expensive than configuration checking. Existing edge checkers either 1) discretize finely and call many configuration checks, or 2) construct a representation of the swept volume as the robot moves along a configuration space path. Both are computationally expensive. Lazy approaches add edges to the roadmap without checking, and remove them after failing a collision test. However, unlike prior approaches that stop after a first solution is found, we continue searching for progressively shorter paths. The speed gains are even more impressive than in feasible planning, because the fraction of irrelevant edges grows toward 1 as the current best path approaches the optimal one.

Another motion planner, FMT*, bears some resemblance to this approach. It first builds a roadmap of feasible samples and computes a cost-to-come heuristic; it then expands a tree forward along the roadmap using the heuristic [10]. The differences are that 1) FMT* is not any-time, and 2) it checks edges along a tree whereas our approach checks edges along a path. This results in many fewer collision checks.

The lazy strategy can be applied to several underlying planners, leading to the Lazy-PRM* and Lazy-RRG* algorithms presented here. It is also compatible with various heuristics, such as bidirectional planning and ellipsoidal pruning [1]. Experiments suggest that the effect of these heuristics is small compared to the overall effect of laziness.

III. Lazy Asymptotically-Optimal Roadmap Methods

This section presents lazy versions of the RRG* and PRM* algorithms, Lazy-RRG* and Lazy-PRM*, and describes important implementation details. It also presents some variants of the basic underlying techniques for bidirectional search, and near-optimal search.

A. Terminology and Assumptions

First let us define some terminology. Let $\mathcal{C}$ be a d-dimensional configuration space, usually taken to be a subset of $\mathbb{R}^d$, and let $\mathcal{F} \subseteq \mathcal{C}$ be the free space (or free subset). The optimal motion planning problem is to find a continuous, minimum cost path $y(u) : [0,1] \rightarrow \mathcal{F}$ connecting a start configuration q_s to a goal set $G \subseteq \mathcal{F}$, i.e., such that $y(0) = q_s$ and $y(1) \in G$. In this paper we will treat path length as the cost function, although the techniques presented herein are largely applicable to other cost functions.

As standard in sampling-based motion planning, $\mathcal{F}$ is described by the *configuration test* $IsFeasible?(q)$ that returns true iff $q \in \mathcal{F}$, and the edge test $IsVisible?(q, q')$ that return true iff the straight-line path from q to q' is contained entirely in $\mathcal{F}$. Furthermore, the algorithm is furnished a *distance metric* $d(q, q')$ on $\mathcal{C}$, and a sampling procedure $Sample()$ which generates a configuration from $\mathcal{F}$ at random according to a probability distribution with strictly positive density everywhere. Throughout this paper, $IsVisible(q, q')$ is implemented by a recursive bisection technique up to a given distance threshold ϵ (noted in the experiments section), and $Sample$ is implemented via rejection sampling from a uniform distribution over $\mathcal{C}$.

In the point-to-point case where the goal G is a singleton configuration $G = \{q_g\}$, it is first added to the roadmap. In the point-to-set case, it is assumed that there is a nonzero probability of sampling a configuration in G using $Sample()$.

B. Summary

The lazy approach incrementally builds two roadmaps, $\mathcal{G} = (V, E)$ and $\mathcal{G}_{lazy} = (V, E_{lazy})$ on the same set of milestones $V \subset \mathcal{F}$, where E contains all edges that are shown to be feasible, and E_{lazy} contains E as well as all *candidate edges* that are not yet shown to be infeasible. It is apparent that if $\mathcal{G}$ contains a path from q_s to G then the path corresponds to a feasible path in $\mathcal{F}$, but this does not necessarily hold with $\mathcal{G}_{lazy}$.

Note that a graph ought to be used, rather than a tree (as in RRT*), because lazy updates frequently revise shortest paths. For this rewiring step, the feasibility status of edges not currently on the shortest path tree can be cached when using a graph. Using a tree, these edges would need to be re-tested for feasibility.

Starting from the start configuration, the planner (Alg. 1) proceeds to repeatedly alternate expansion and lazy update steps. The $LazyExpand$ subroutine expands $\mathcal{G}_{lazy}$ using some *exploration strategy*. These strategies are essentially the same as those in standard sampling-based planners, except that edges are not checked for collision. Slightly different implementations yield the Lazy-PRM* and Lazy-RRG* algorithms.

Algorithm 1 Lazy-Planner(q_s, G, N)
```
1: Initialize $\mathcal{G}$ and $\mathcal{G}_{lazy}$ to start configuration
2: $C_{best} \leftarrow \infty$                    ▷ Cost of best path found
3: for $i = 1, 2, \ldots, N$ do
4:     Call $LazyExpand(\mathcal{G}_{lazy})$
5:     Call $LazyUpdate(\mathcal{G}, \mathcal{G}_{lazy}, C_{best})$
   return $ShortestPath(\mathcal{G}, q_s, G)$
```

The key contribution of this paper is the $LazyUpdate$ subroutine, which finds candidate cost-improving paths in $\mathcal{G}_{lazy}$ and checks the edges along them for collision. If an edge is feasible, it is added to $\mathcal{G}$. If it is infeasible, it is removed from $\mathcal{G}_{lazy}$. This requires a bit of additional overhead to compute candidate paths, which is done efficiently using a dynamic shortest paths algorithm.

Typically, the planner is run in any-time mode where the main loop is terminated whenever a solution is desired (usually based on a time limit) and the planner returns the best solution found so far. For point-to-point planning problems, we modify line 1 to add the goal q_g to $\mathcal{G}$ and $\mathcal{G}_{lazy}$ upon initialization.

C. Implementation details

We shall now describe the implementation of the key phases of the planner, $LazyExpand$ and $LazyUpdate$.

Expansion is essentially the same as in other sampling-based asymptotically-optimal planners, except only vertices are checked for collision, not edges. The pseudocode for Lazy-PRM* is given in Alg. 2, while the psudocode for Lazy-RRG* is given in Alg. 3

Algorithm 2 $LazyExpandPRM^*(R)$
```
1: $q \leftarrow Sample()$
2: if $IsFeasible?(q)$ then
3:     Add vertex $q$ to $R$.
4:     for all $v \in Neighbors(R, q, i)$ do
5:         Add edge $(q, v)$ to $R$
```

Here, $Neighbors(R, q, i)$ is a set of "close" neighbors in the roadmap R that will be connected to q, on the i'th step of Lazy-Planner. As in [11], we may return either the k-nearest

Algorithm 3 $LazyExpandRRG^*(R)$
```
1: $q_{rand} \leftarrow Sample()$
2: $q_{near} \leftarrow Nearest(q_{rand}, R)$
3: $q \leftarrow q_{near} + max(\frac{r(i)}{d(q_{near}, q_{rand})}, 1)(q_{rand} - q_{near})$
4: if $IsFeasible?(q)$ then
5:     Add vertex $q$ to $R$.
6:     Add edge $(q_{near}, q)$ to $R$
7:     for all $v \in Neighbors(R, q, i)$ do
8:         Add edge $(q, v)$ to $R$
```

neighbors or a r-ball selection strategy, with the connection factors $k(i) = O(\log(i))$ and $r(i) = O((\log(i)/i)^{1/d})$ that ensure that the shortest path in $\mathcal{G}$ converges to the shortest path in $\mathcal{F}$. The $Nearest$ subroutine in Lazy-RRG* searches for the closest node in R that is connected to the start q_s (unlike non-lazy RRG*, the roadmap may become disconnected).

In $LazyUpdate$ the planner checks shortest candidate paths in $\mathcal{G}_{lazy}$ for feasibility, adding edges to $\mathcal{G}$ when found feasible, and deleting them from $\mathcal{G}_{lazy}$ if found infeasible. We first demonstrate the principle using a simple but slow implementation in Alg. 4. Let the expression $C(p)$ yield the cost of the path p, and let $ShortestPath(R, q_s, G)$ yields a minimum cost path in the roadmap R from the start configuration q_s to any configuration in the goal set G.

Algorithm 4 $LazyUpdate1(\mathcal{G}, \mathcal{G}_{lazy}, C_{best})$
```
1: repeat
2:     $p \leftarrow ShortestPath(\mathcal{G}_{lazy}, q_s, G)$
3:     if $C(p) < C_{best}$ then              ▷ Shorter path found
4:         for all edges $(u, v) \in p$ not already in $\mathcal{G}$ do
5:             if $IsVisible?(u, v)$ then
6:                 Add edge $(u, v)$ to $\mathcal{G}$
7:             else
8:                 Delete edge $(u, v)$ from $\mathcal{G}_{lazy}$
9:                 Return to Line 2
10:        if all edges in $p$ are in $\mathcal{G}$ then
11:            Set $C_{best} \leftarrow C(p)$       ▷ $p$ is new best path
12: until $C_{best} = C(p)$
```

We remark that $LazyUpdate$ maintains the invariant that the shortest path in $\mathcal{G}$ is identical to the shortest path in $\mathcal{G}_{lazy}$ and has cost C_{best}.

Note that this implementation computes a new shortest path in every inner loop. This can be quite expensive because it requires solving a search problem, e.g., using Dijkstra's algorithm, which runs in time $O(|E|log|V|)$. Moreover, in pathological cases the number of shortest path queries requested over the duration of planning could be up to $O(|E|)$. The next section presents a method that avoids much of this overhead.

D. Dynamic Shortest Path Updates

To avoid calling multiple shortest path queries, we use a dynamic shortest paths algorithm [6]. Such algorithms maintain the cost $c[v]$ to every vertex as well as each vertex's predecessor $\pi[v]$ in a shortest paths search on a graph. The arrays c and π are updated dynamically every time a vertex is added to the graph, an edge is added to the graph, or an edge is removed from the graph. We note that the RRT* planner is simply RRG* where only the predecessor $\pi[v]$ is stored; the RRT* "rewiring" step performs operations identical to those of a dynamic shortest paths algorithm when an edge is added. Lazy-RRG* additionally has functionality to update costs and parents when an edge is deleted.

We maintain the arrays storing cost c and predecessor π for the roadmap $\mathcal{G}_{lazy}$. We also maintain a list of goal milestones Q_G, which is updated whenever a new configuration is added to the roadmap. The new $LazyUpdate$ method is listed in Alg. 5. Here, $PathTo$ follows backpointers along π to yield the shortest path. A small optimization pre-sorts the vertices in $Neighbors$ by increasing $c[v] + d(v, q)$, which eliminates many shortest-path updates as noted by [15].

Algorithm 5 $LazyUpdate2(\mathcal{G}, \mathcal{G}_{lazy}, C_{best})$

```
1:  repeat
2:      $q_g \leftarrow \arg\min_{q \in Q_G} c[q]$
3:      if $c[q_g] < C_{best}$ then                ▷ Shorter path found
4:          $p \leftarrow PathTo(q_g)$
5:          for all edges $(u, v) \in p$ not already in $\mathcal{G}$ do
6:              if $IsVisible?(u, v)$ then
7:                  Add edge $(u, v)$ to $\mathcal{G}$
8:              else
9:                  Delete edge $(u, v)$ from $\mathcal{G}_{lazy}$
10:                 Return to Line 2
11:         if all edges in $p$ are in $\mathcal{G}$ then
12:             Set $C_{best} \leftarrow c[q_g]$     ▷ Path to $q_g$ is new best
13: until $q_g = nil$ or $C_{best} = c[q_g]$
```

Dynamic shortest paths updates are performed after the following graph operations:

- After adding a vertex v (Line 3 of Alg. 2 and Line 5 of Alg. 3): set $c[v] \leftarrow \infty$ and $\pi[v] \leftarrow nil$
- After adding an edge $u \rightarrow v$ (Line 5 of Alg. 2 and Lines 6 and 8 of Alg. 3): if the edge yields a shorter path to v, i.e., $c[u] + d(u, v) < c[v]$, then $c[v]$ and $\pi[v]$ are updated to use u as a parent, and the decreases in cost are propagated downstream from v. Note that since we are assuming an undirected graph, this operation is performed twice, once in each direction.
- After deleting an edge $u \rightarrow v$ (Line 9 of Alg. 5): if $\pi[v] \neq u$, then nothing changes because the edge was not a part of a shortest path through v. Otherwise, we search through alternate parents of v to find a least-cost parent. We then set this to the parent of v and propagate the increased cost downstream. If there are no other alternatives, we set $\pi[v]$ and $c[v] = \infty$.

As another optimization, we noticed that the planner often wastes time in the following "optimistic thrashing" sequence, which occurs when a configuration q that jumps across obstacles is optimistically added to the roadmap:

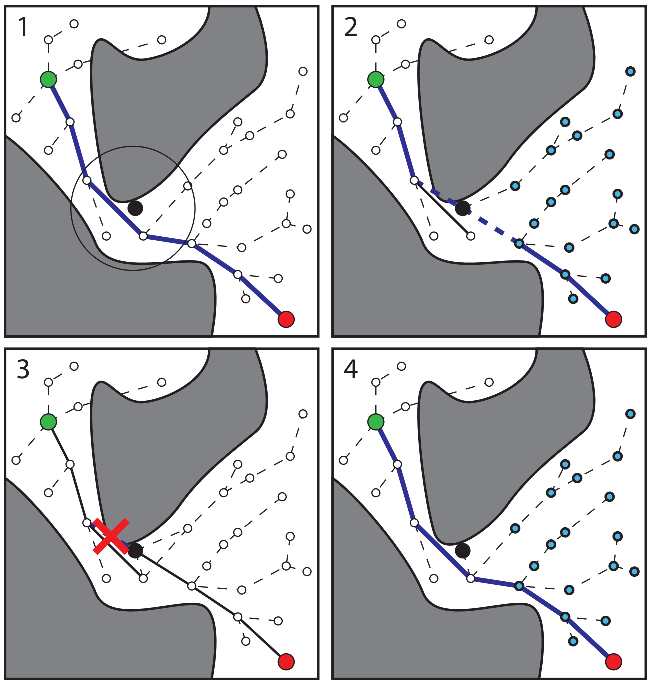

Fig. 2. Illustrating the "optimistic thrashing" phenomenon. (1) A new milestone is sampled (black circle) and candidate edges are added to all nodes within a given radius. (2) This introduces a new optimal candidate path to the goal (highlighted edges), and the planner would optimistically update shortest paths (dotted lines) to the shaded nodes, which comprise about half of the roadmap. (3) The edges along the new path are checked and found to be infeasible, and (4) costs are reverted to their original values. Our implementation pre-checks the highlighted edges *before* updating shortest paths to avoid thrashing.

1) q is connected to neighbors in the roadmap, which updates shortest paths to a (possibly huge) fraction of the roadmap due to the optimistic cost estimate.
2) A new candidate optimal path is found to the goal.
3) This path is checked and found to be infeasible on an edge leading out of q, and hence shortest paths are reverted back to their original values.

As a result, if the planner can quickly determine that a new candidate edge ending at q would create a new best path to the goal, then it would make sense to pre-check the two edges through q immediately rather than wasting time performing step 1 and undoing it all in step 3. To do so, we actually store two shortest paths data structures, one forward from the start (c_{fwd}, π_{fwd}) and one backward from the goal (c_{bwd}, π_{bwd}). Doing so adds the expense of essentially doubling the shortest paths bookkeeping, but we usually save the huge expense of optimistic thrashing.

E. Further Pruning

Our planner also contains several pruning steps to achieve further reductions in computational cost. (In our experiments, we apply all of these enhancements to both lazy and non-lazy planners to ensure a fair comparison.)

We prune unnecessary vertices: when considering a new milestone q, if $d(q_s, q) + h(q) \geq C_{best}$, where $h(q)$ is a heuristic function that underestimates the cost to goal, then no optimal path can pass through q and it can be pruned. For example, for point-to-point planning we let $h(q) = d(q, q_g)$, this yields the common ellipsoidal pruning strategy [1], which helps the most when the optimal path is close to a straight line and/or short compared to the diameter of the space. A slight improvement to this strategy uses $c[q]$ in place of $d(q_s, q)$.

We prune unnecessary edges: as in LBT-RRT*, if the cost of a new candidate path to a node on $\mathcal{G}_{lazy}$ is greater the cost on the actual graph $\mathcal{G}$, then the edge can be pruned. That is, if $c[u] + d(u, v) \geq c_{actual}[v]$, then the edge (u, v) does not need to be added.

We prune shortest path updates: search is halted beyond nodes where $c[v] \geq C_{best}$ holds. This reduces overhead when the search explores too widely early on, before a short path to the goal is found.

F. Computational Complexity

An expression for the cost of the planner is

$$c_{NN} + c_{SP} + c_{VC} + c_{EC} \tag{1}$$

where c_{NN} is the total cost of performing neighborhood queries, c_{SP} is the cost of shortest-path solving, c_{VC} is the cost of vertex collision checking, and c_{EC} is the cost of edge collision checking. Since c_{NN} and $c_{VC}$are unchanged by lazy approaches, we will study c_{SP} and c_{EC} in this section.

With N milestones, the number of edges in the roadmap is $O(N \log N)$ since RRT* / RRG* connection neighborhoods contain $O(\log N)$ vertices. In non-lazy planning, c_{SP} is $O(N \log^2 N)$, while c_{EC} is $O(N \log N)$. But this expression hides a typically large, problem-dependent constant factor so that $c_{EC} >> c_{SP}$ for most reasonable values of N (Fig. 3, top). In lazy planning, c_{EC} starts approximately $O(N)$ but asymptotically approaches a constant. It is difficult to give a expression of c_{SP} because the analysis of dynamic shortest paths is challenging [6]. We do know it is lower bounded by $O(N \log^2 N)$ and upper bounded by the cost of a naïve implementation of $LazyUpdate1$, $O(N^2 \log^2 N)$. In practice, c_{SP} appears to take a constant fraction of running time (Fig. 3, bottom), suggesting that empirical performance is closer to the lower bound.

G. Asymptotic Optimality Arguments

We informally argue the key steps in proving asymptotic optimality of Lazy-PRM*. It has three steps:

1) For any given N, the milestones in $\mathcal{G}_{lazy}$ in Lazy-PRM* are the same as those of PRM*, assuming the same sampling sequence.
2) The shortest feasible path in $\mathcal{G}_{lazy}$ will be added to $\mathcal{G}$ after $LazyUpdate$ is called. Hence, the shortest path is the same as the one produced by PRM*.
3) Since PRM* is asymptotically optimal [11], Lazy-PRM* is optimal as well.

It is less obvious to prove asymptotic optimality of Lazy-RRG*, because it does not add the same milestones to the roadmap as RRG*. But, the above argument should hold because Lazy-RRG* behaves identically to Lazy-PRM* in the dense limit where no partial extensions are made (i.e., $q = q_{rand}$ in Step 3 of Alg. 3).

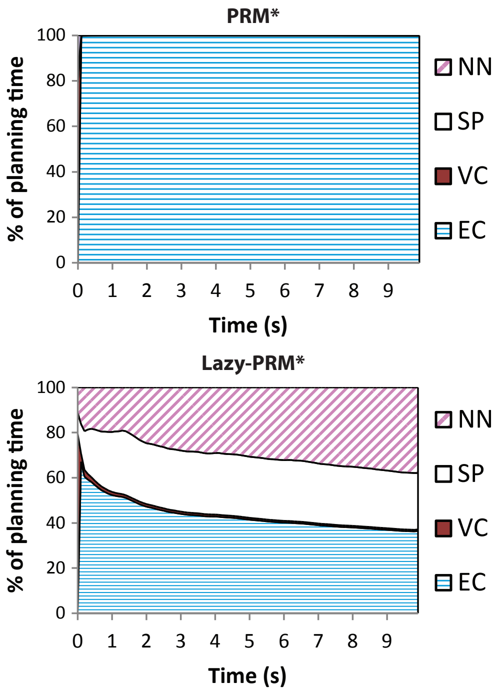

Fig. 3. The breakdown of time used by various components of PRM* and Lazy-PRM*. Values are averaged over 10 runs on the example of Fig. 4.

IV. Experiments

Here we conduct comparisons of lazy and non-lazy versions of PRM* and RRT* [11], as well as LBT-RRT* [15], a near-optimal planner that reduces the number of edge collision checks by sacrificing optimality. Suboptimality factors of 0.1 and 0.2 are tested. All experiments are conducted on a 2.67GHz Intel Core i7 PC, running on a single thread. Planners are implemented in C++. To ensure a fair comparison, all lazy and non-lazy planners are implemented using identical subroutines throughout, e.g., collision detection, graph maintenance, nearest neighbors queries, etc.

A. Rigid Planar Bar

The first experiments are conducted on a planar workspace with a rectangular robot that can translate and rotate (Figs. 4–5). Edge collision detection is performed via recursive bisection up to the tolerance $\epsilon = 0.001$. Each planner is run 10 times with different random seeds, and curves plot average performance after at least half of the runs succeeded. We plot the cost of the best path found so far at each point in time (middle), and the number of edge collision checks against the number of planning iterations (right). Fig. 5 shows a narrow passage problem in which lazy approaches vastly outperform the non-lazy ones, and furthermore lead to an asymptotically constant number of edge checks.

Fig. 5 shows a pathological problem in which lazy planning performs poorly at first, because the lazy roadmap

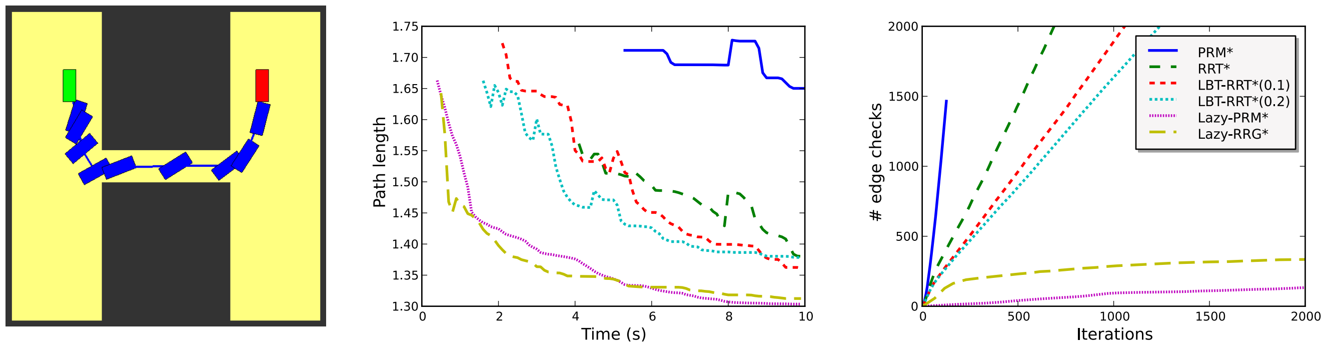

Fig. 4. Rigid bar narrow passage problem. Middle: convergence of the optimal feasible path over time. Right: cumulative number of edge checks over planning iterations. Lazy approaches exhibit a sub-linear number of edge checks.

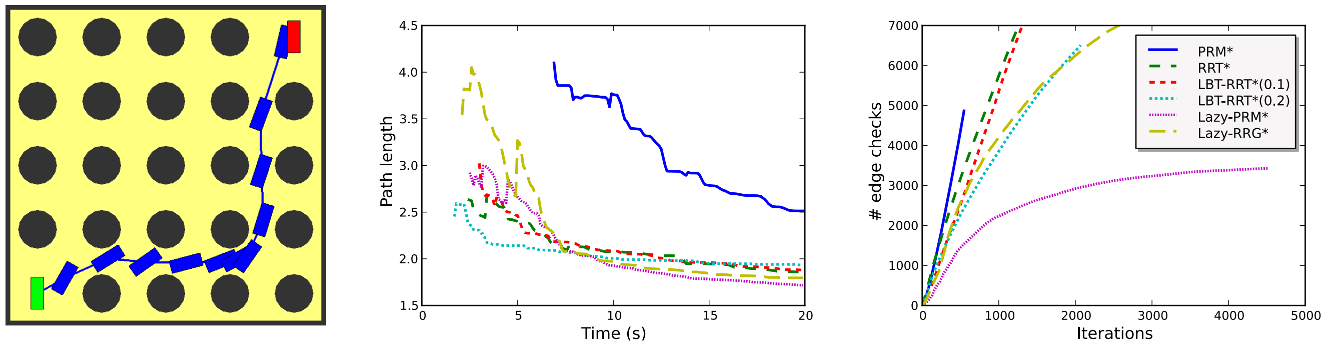

Fig. 5. Pathological rigid bar problem, with small obstacles and many homotopy classes. Lazy approaches perform worse at first, but at approximately 7 s they start to perform better than the other methods due to a faster convergence rate.

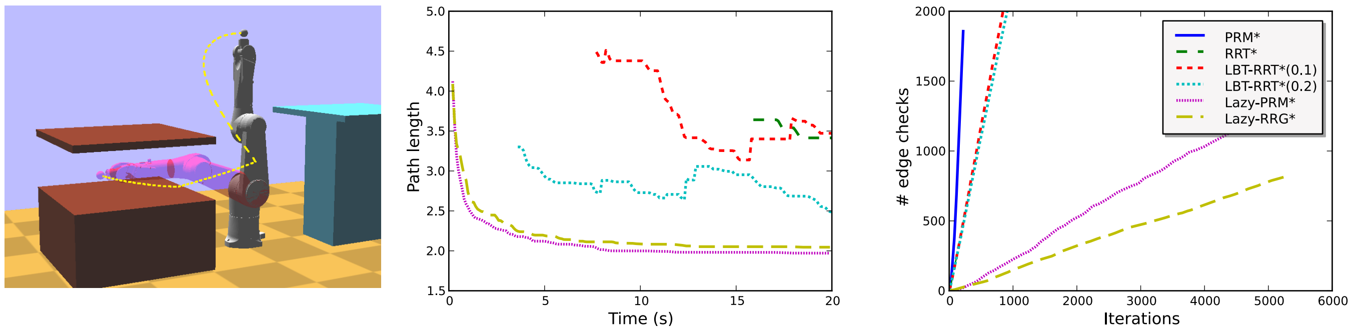

Fig. 6. A 6D manipulator planning problem on the Staubli TX90L. The start configuration is drawn in grey and the goal is drawn in red.

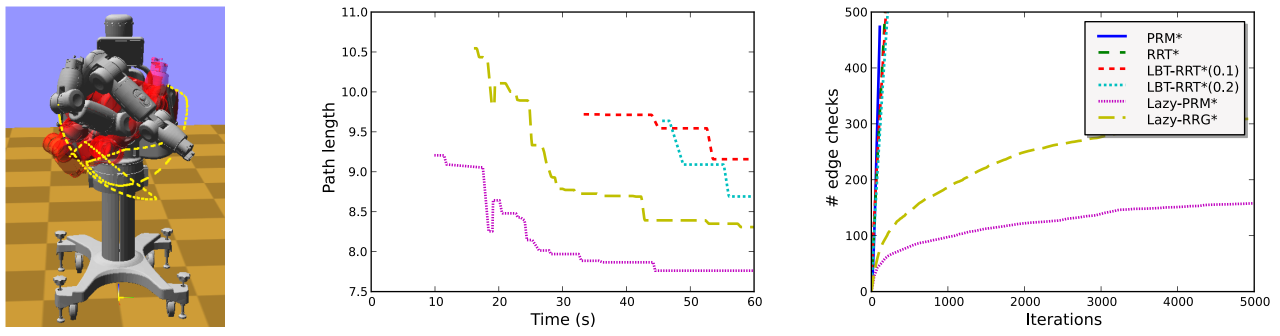

Fig. 7. A 12D problem on the Rethink Robotics Baxter in a tangled situation. The start configuration is drawn in grey and the goal is drawn in red.

makes many optimistic jumps over obstacles and must therefore perform extensive revisions to the roadmap. However, the lazy approaches eventually close the gap in performance by performing fewer edge checks, and overtake the non-lazy approaches near the 10 s mark.

B. Robot Manipulators

Next, we test a 6D problem on the TX90L industrial manipulator (Fig. 6), and a 12D problem on the Rethink Robotics Baxter two-armed manipulator (Fig. 6). In both cases, configuration collision detection is performed on the raw CAD models (68,862 and 397,830 triangles, respectively) using the PQP bounding volume hierarchy method [8]. Edge collision detection is performed with tolerance $\epsilon = 0.01\,rad\ L_\infty$ distance between configurations.

Since collision detection is much more costly, the benefits of laziness are even more striking. In the 6D case, lazy approaches are orders of magnitude faster than RRT* for the same level of quality, and PRM* does not even find a solution within 20 s of planning. In the 12D case, neither RRT* nor PRM* finds a solution within 60 s of planning.

V. Conclusion

This paper presented a lazy collision checking strategy for sampling-based optimal motion planning. The strategy avoids checking edges that have no chance to improve upon the current best path, thereby reducing the per-sample computational cost and accelerating convergence. New methods are presented for reducing the overhead of shortest path queries. The approach is applied to the RRG* and PRM* planners, with lazy planners are more readily applicable to high-dimensional spaces and less sensitive to collision detection cost than their non-lazy versions.

Lazy planning is not always superior; in pathological cases with small obstacles, they can perform worse at first, but in the long term, they overtake non-lazy versions due to a superior convergence rate. Future work may consider mitigating this problem by immediately checking edges longer than a given threshold (e.g., typical C-obstacle widths); such a threshold could be learned adaptively during planning. Other work might push the lazy paradigm farther. For example, full collision checking is unnecessary for milestones not along an optimal path. An even lazier method might begin by only partially checking configurations, which would guide the checking of full configurations, and finally the checking of paths.

Acknowledgment

The author would like to thank Oren Salzman for stimulating discussions about this work and the anonymous reviewers for their helpful suggestions in improving this manuscript.

References

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

[2] R. Alterovitz, S. Patil, and A. Derbakova. Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning. In *IEEE Int. Conf. Rob. Aut.*, pages 3706–3712, 2011.

[3] J. Bialkowski, M. Otte, S. Karaman, and E. Frazzoli. Efficient collision checking in sampling-based motion planning. In *Alg. Found. Robotics X*, volume 69. Springer, Berlin / Heidelberg, 2013.

[4] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *IEEE Int. Conf. Rob. Aut.*, pages 521–528, San Francisco, CA, 2000.

[5] A. Dobson and K. E. Bekris. Sparse roadmap spanners for asymptotically near-optimal motion planning. *Int. J. Rob. Res.*, 33(1):18–47, 2014.

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

[7] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *IEEE/RSJ Int. Conf. Intel. Rob. Sys.*, Chicago, USA, 2014.

[8] S. Gottschalk, M. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. In *ACM SIGGRAPH*, pages 171–180, 1996.

[9] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *IEEE Int. Conf. Rob. Aut.*, pages 2219–2226, 1997.

[10] L. Janson and M. Pavone. Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions. In *Intl. Symp. Robotics Research*, 2013.

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

[12] L. E. Kavraki, P. Svetska, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. and Autom.*, 12(4):566–580, 1996.

[13] S. M. LaValle and J. J. Kuffner, Jr. Randomized kinodynamic planning. *Int. J. Rob. Res.*, 20(5):379 – 400, 2001.

[14] J. Luo and K. Hauser. An empirical study of optimal motion planning. In *IEEE/RSJ Int. Conf. Intel. Rob. Sys.*, Chicago, USA, 2014.

[15] O. Salzman and D. Halperin. Asymptotically near-optimal rrt for fast, high-quality, motion planning. In *IEEE Int. Conf. Rob. Aut.*, Hong Kong, 2014.

[16] G. Sánchez and J.-C. Latombe. On delaying collision checking in PRM planning: Application to multi-robot coordination. *Int. J. of Rob. Res.*, 21(1):5–26, 2002.