The Provable Virtue of Laziness in Motion Planning*

Nika Haghtalab[1] , **Simon Mackenzie**[2] ,
Ariel D. Procaccia[3] , **Oren Salzman**[3] and **Siddhartha Srinivasa**[4]
[1]Microsoft Research
[2]University of New South Wales
[3]Carnegie Mellon University
[4]University of Washington

Abstract

The *Lazy Shortest Path (LazySP)* class consists of motion-planning algorithms that only evaluate edges along candidate shortest paths between the source and target. These algorithms were designed to minimize the number of edge evaluations in settings where edge evaluation dominates the running time of the algorithm such as manipulation in cluttered environments and planning for robots in surgical settings; but how close to optimal are LazySP algorithms in terms of this objective? Our main result is an analytical upper bound, in a probabilistic model, on the number of edge evaluations required by LazySP algorithms; a matching lower bound shows that these algorithms are asymptotically optimal in the worst case.

1 Introduction

The simplest motion planning model [Halperin *et al.*, 2017; LaValle, 2006] involves a robot system R moving in a workspace $W \in \{\mathbb{R}^2, \mathbb{R}^3\}$ cluttered with obstacles O. Given an initial placement s and a target placement t of R, we wish to determine whether there exists a collision-free motion of R connecting s and t, and, if so, to plan such a motion.

Typically, R is abstracted as a point, or a *configuration*, in a high-dimensional space called the *configuration space* X, where each configuration maps R to a specific placement in W [Lozano-Perez, 1983]. The configuration space is subdivided into the free and forbidden spaces, corresponding to placements of R that are free or that intersect with an obstacle, respectively. Since the general motion-planning problem is PSPACE-hard [Hopcroft *et al.*, 1984], a common approach is to use sampling-based algorithms [Kavraki *et al.*, 1996; Hsu *et al.*, 1999; LaValle and Kuffner, 1999; Karaman and Frazzoli, 2011]. These algorithms approximate X via a discrete graph G called a *roadmap*. Vertices in G correspond to sampled configurations in X, and edges in G correspond to local paths (typically straight lines). Approximately solving the motion-planning problem thus reduces to the problem of finding a collision-free shortest path in G between the vertices corresponding to s and t.

Testing if a vertex or an edge of G is collision free requires one or more geometric tests called *collision detection*. Arguably, collision detection in general, and edge evaluation in particular, are the most time-consuming operations in sampling-based algorithms [LaValle, 2006; Choset *et al.*, 2005]. Thus, path planning on G differs from traditional search algorithms such as Dijkstra [1959] or A* [Hart *et al.*, 1968], where the graph is typically implicit and large, but edge evaluation is trivial compared to search. Indeed, much recent work in motion planning focuses on evaluating the edges of G *lazily*, that is, assuming that the edges do not intersect with the obstacles O [Bohlin and Kavraki, 2000; Hauser, 2015; Dellin and Srinivasa, 2016; Salzman and Halperin, 2015; Choudhury *et al.*, 2017; Mandalika *et al.*, 2018; Mandalika *et al.*, 2019].

In a recent paper, Dellin and Srinivasa [2016] present a unifying formalism for shortest-path problems where edge evaluation dominates the running time of the algorithm. Specifically, they define and investigate a class of algorithms termed *Lazy Shortest Path (LazySP)*, which run any shortest-path algorithm on G followed by evaluating the edges along that shortest path. The algorithms are differentiated by an *edge selector* function, which chooses the edges the algorithm evaluates along the shortest path. Dellin and Srinivasa show that several prominent motion-planning algorithms are captured by LazySP, using a suitable choice of this selector. Furthermore, they evaluate the algorithm empirically on a wide range of edge selectors and scenarios and show that, using this approach, nontrivial problems can be solved within seconds.

LazySP was proposed as an algorithm that attempts to minimize the overall number of edges evaluated (or *queried*) in the process of solving a given motion-planning problem. A natural question to ask is

> *... what is the query complexity of LazySP, and is its query complexity the best possible?*

In other words, can we bound the number of edges evaluated by LazySP as a function of the complexity of the roadmap G? And are there algorithms not in this class that have lower query complexity?

To address these questions, we need to explicitly model how queries are answered. We start in Section 3 by considering the *deterministic* setting, where the set of collision-free

*See the full version of this paper [Haghtalab *et al.*, 2018] for proofs and additional material.

edges is determined upfront. Our first result establishes that, in this model, it is optimal to always test edges along the shortest path, i.e., in every instance there is an edge selector for which LazySP is optimal. Although the edge selector in question requires full access to the set of collision-free edges, so the real-world implications of this result are limited, it does provide a theoretical underpinning for the idea of restricting queries to shortest paths, which lies at the heart of LazySP.

In practice, we are interested in a slightly more complex model, which we call the *probabilistic* setting. Here, each edge is endowed with a probability of being in collision — a common assumption in motion planning (see, e.g., Choudhury *et al.* 2016)—and we are interested in policies that minimize the query complexity, that is, policies that minimize the *expected* number of steps until the algorithm finds the shortest path or declares that no path exists. We first show that there are instances where LazySP is suboptimal, regardless of the edge selector. In a nutshell, we describe a delicate construction where initially querying edges that are not on the shortest path provides valuable information for subsequent queries.

So, in the probabilistic setting, LazySP is just a proxy for the (presumably intractable) optimal policy, but is it a good proxy? We answer this question in the positive. Our main result is that the query complexity of LazySP (with an edge selector satisfying a certain *connectivity* property) is bounded by $O(n/p)$ edge evaluations with high probability, where n is the number of vertices in G, and p is the minimum probability on any edge. We complement this result with an $\Omega(n/p)$ lower bound that holds for *every* algorithm that is guaranteed to be correct. We conclude that, from a worst-case viewpoint, LazySP is, in fact, (asymptotically) optimal.

2 Model

An instance of our problem is given by a multigraph[1] $G = (V, E)$ whose set of vertices includes a source vertex s and a target vertex t. We say that a graph $G' = (V, E')$ is a subgraph of G if $E' \subseteq E$. Given a graph $G = (V, E)$ and a subgraph $G' = (V, E')$ of G, an oracle $\mathcal{O}^G_{G'}$ is a function that takes as input an edge $e \in E$ and returns YES if $e \in E'$, and NO otherwise. When G is clear from the context, we suppress it in this notation.

In the *path-finding* problem, an algorithm ALG is given a graph G and an oracle $\mathcal{O}_{G'}$. The goal of the algorithm is to find the shortest s-t path in G'. Since G' is not revealed to the algorithm directly, the algorithm has to query $\mathcal{O}_{G'}$ on specific edges of G to find a path. That is, $\text{ALG}(G, \mathcal{O}_{G'})$ issues a sequence of edge queries to $\mathcal{O}_{G'}$, and upon termination, returns an s-t path or decides that none exists. To ground this model in the context of a motion-planning algorithm, the graph G is lazily constructed and can have edges that are in collision while the subgraph G' contains only collision-free edges.

For an algorithm to be *correct*, we require that it correctly identifies a shortest s-t path in G', or that it *certify* that none exists (by invalidating every possible path), for any G and $G' \subseteq G$. Therefore, a correct algorithm can only terminate when the solution it provides continues to be correct even if the responses to unqueried edges are selected adversarially. More formally, let $Q \subseteq E$ be the set of edges queried by a correct algorithm ALG on G and $\mathcal{O}_{G'}$. Let $Q_y = Q \cap E'$ and $Q_n = Q \setminus E'$ be the set of queried edges that, respectively, belong and do not belong to G'. Then ALG can terminate only if there is a shortest s-t path in G', denoted P^*, such that $P^* \subseteq Q_y$, and there is no s-t path in $(V, E \setminus Q_n)$ that is shorter than P^*. If no path exists, then ALG can terminate only if there is no s-t path in $(V, E \setminus Q_n)$.

Algorithm 1: LAZYSP_f

input: Graph G and oracle $\mathcal{O}_{G'}$

```
Q_n ← ∅    // in-collision evaluated edges
Q_y ← ∅ // collision-free evaluated edges
while there exists[2] a shortest s-t path P in E \ Q_n do
    if P ⊆ Q_y then return P
    e ← f(P, Q_y, Q_n) // select edge along P
    if O_G'(e) = YES then Q_y ← Q_y ∪ {e}
    else Q_n ← Q_n ∪ {e}
end
return ∅;
```

Clearly, an algorithm that first queries all edges in E, thereby fully constructing G', and only then finds the shortest s-t path, is a correct algorithm. However, such an algorithm may use a large number of queries, some of which may be unnecessary. In this paper, we are interested in algorithms that find a shortest s-t path using a minimal number of queries. We denote the number of queries that ALG makes on input G and $\mathcal{O}_{G'}$ by $\text{cost}(\text{ALG}(G, \mathcal{O}_{G'}))$.

We are especially interested in the *LazySP* class of algorithms, introduced by Dellin and Srinivasa [2016]. Any algorithm in the class *LazySP* is determined by an *edge selector*, which, informally, decides which edge to query on a given s-t path. Formally, let $\mathcal{P}$ be the set of all s-t paths in G. An edge selector is a function $f : \mathcal{P} \times 2^E \times 2^E \to E$ that takes any s-t path $P \in \mathcal{P}$, a subset of queried edges Q_y that are in E', and a subset of queried edges Q_n that are not in E', and returns an edge $e \in P \setminus Q$. For example, a *Forward edge selector* returns the first unqueried edge in P. See [Dellin and Srinivasa, 2016] for examples of edge selectors.

Given an edge selector f, the corresponding $\text{LAZYSP}_f \in$ *LazySP* is described in Algorithm 1. At a high level, LAZYSP_f, in a given time step, considers a candidate *shortest* s-t path P over all those edges whose existence has not yet been ruled out by the oracle. Then, it uses the edge selector to query an unqueried edge $e \in P$. It updates the set of queried edges and repeats. At any point, if the edges of path P that is currently under consideration are all verified, the algorithm terminates and returns P. If no viable s-t paths remain, the algorithm terminates and certifies that no s-t path exists in G'.

Let us conclude this section with an example of the execution of LAZYSP with the forward edge selector, which also

[1] We deal with multigraphs, rather than simple graphs, mostly for ease of exposition; see [Haghtalab *et al.*, 2018] for a discussion of this point. We simply refer to G as a *graph* hereinafter.

[2] If there are multiple s-t paths of the same length, the algorithm breaks ties according to a consistent tie-breaking rule.

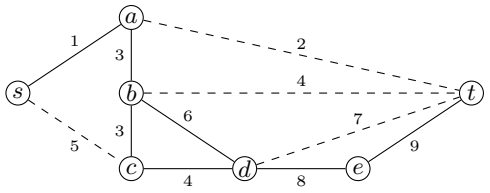

Figure 1: Example of the execution of LAZYSP with the forward edge selector. Solid edges are in E', dashed edges are in $E \setminus E'$.

illustrates some of the terminology introduced earlier. Figure 1 shows the set of vertices $V = \{s, t, a, b, c, d, e\}$ shared by G and G', as well as two types of edges: those in E', shown as solid edges, and those in $E \setminus E'$, shown as dashed edges. The order in which edges are queried is shown as labels on the edges. This order on edge queries is induced by evaluating shortest paths in the following order: sat, $sabt$, $scdt$, $sabdt$, and $sabdet$.

3 The Deterministic Setting

In this section, we consider the problem of using a minimum number of edge queries to find a shortest s-t path, or verifying that no s-t path exists, when a subgraph $G' \subseteq G$ is *deterministically* chosen (but not revealed to the algorithm).

In more detail, let $G = (V, E)$ be a graph, and let $G' = (V, E')$ be a subgraph of G. Recall that $\text{cost}(\text{ALG}(G, \mathcal{O}_{G'}))$ denotes the number of edge queries ALG makes on graph G when oracle responses are according to graph G'. Our first result asserts that the class *LazySP* is optimal in this setting, in the sense that for any correct algorithm there is a LAZYSP algorithm (with a specific edge selector) that finds the shortest path using at most as many queries.

Theorem 1. *For any graph G and $G' \subseteq G$, and any correct algorithm* ALG*, there exists* $\text{ALG}' \in$ *LazySP such that*

$$\text{cost}(\text{ALG}'(G, \mathcal{O}_{G'})) \leq \text{cost}(\text{ALG}(G, \mathcal{O}_{G'})).$$

We can alternatively interpret Theorem 1 in a model where LAZYSP may be equipped with an *omniscient* edge selector that has full access to G'. In particular, this omniscient edge selector can compute Q^*, which, by the way, requires solving an NP-hard variant of SET COVER. Even though the algorithm already knows G', it still has to issue queries as it must *certify* that P^* is indeed the shortest path (if an s-t path exists).

Clearly, an omniscient edge selector is impractical. The significance of Theorem 1, therefore, is mostly conceptual. It suggests that the restriction that algorithms must always query edges on the current shortest path is not a barrier to optimality. This gives theoretical justification for the *LazySP* class. However, as we shall see shortly, the message is more nuanced when the outcomes of queries are randomized.

4 The Probabilistic Setting

Let $p \in (0, 1)$ be the probability that any given edge in G exists in G'. We denote by $G' \sim_p G$ the process of generating a random graph $G' = (V, E')$ from G by allowing each $e \in E$ to belong to E' with probability p, independently. We suppress p in this notation when it is clear from the context.

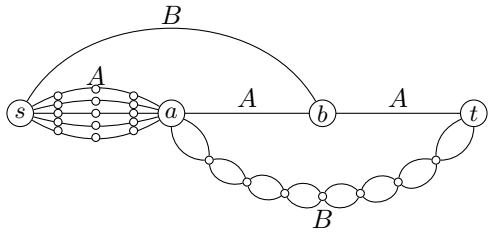

Figure 2: A graph for which no algorithm in *LazySP* is an optimal query policy. All arcs labeled by A and B include multi-edge structures shown in Figures 3 and 4, respectively. For clarity, we include two examples of these structures on sa and at in this figure.

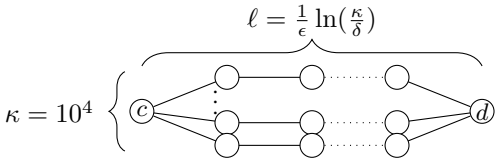

Figure 3: Structure A used on arcs sa, ab, and bt in Figure 2. We refer to one path connecting c and d as a "string".

Here, a subgraph $G' = (V, E') \sim_p G$ is realized according to edge probability p, but it is not revealed to the algorithm. As before, the algorithm receives G and $\mathcal{O}_{G'}$ as input, and uses $\mathcal{O}_{G'}$ to verify whether an edge exists. The goal of the algorithm is to minimize the *expected* number of edge queries over $G' \sim_p G$, such that it *correctly* either (i) returns a path that is the shortest s-t path in G', or (ii) certifies that there is no s-t path in G'. Note that, although the expected number of queries an algorithm issues is taken over $G' \sim G$, the correctness condition must hold for *every* G'.

4.1 Suboptimality of LazySP

Our next result asserts that the class of algorithms *LazySP* does not always include an optimal query policy, which minimizes the expected number of queries. At a high level, the reason behind this is that, in some graphs, querying a few edges that are not on the shortest path can identify the most important regions of the graph, which should be explored next. To see this, consider the graph in Figure 2. In this graph, the arcs marked by A and B each include multi-edge structures shown in Figures 3 and 4, respectively. Structures A and B are designed so that arcs labeled by B are much longer than A, so any LAZYSP algorithm starts by querying the arcs labeled by A.

We compare the cost of any LAZYSP $\in$ *LazySP* (for an arbitrary edge selector) to that of an algorithm ALG defined as follows. ALG first queries all the edges in the multi-edge structures B on arcs sb and at. There are two cases:

1. A path exists in both of the structures sb and at, or in neither one: In this case, ALG calls LAZYSP on the original graph.
2. There is a path in exactly one of the sb or at structures: Without loss of generality (by symmetry) assume that at has a path. Then, ALG queries the edges in structure A on sa, ab and bt in order, until it verifies that at least one of these structures does not have a path or all do.

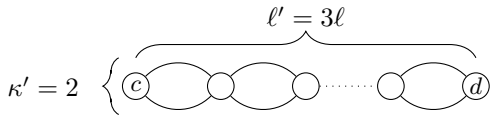

Figure 4: Structure B used on arcs sb and at in Figure 2.

Then, it returns the shortest s-t path on the edges whose existence has been verified by the queries, or certifies that no s-t path exists.

It is not hard to see that ALG demonstrates the required guarantees for a correct algorithm, i.e., upon its termination it correctly certifies that there is no s-t path or returns the shortest s-t path in the realized graph.

Let us provide an overview of why ALG queries fewer edges than any LAZYSP algorithm in expectation. The structures A and B are designed so that structure A requires more queries than structure B. Additionally, structure A almost certainly fails to have a path, while structure B has a path with a probability close to $\frac{1}{2}$. Note that such a graph almost certainly does not have a path, so a large fraction of $\mathbb{E}[\text{cost}(\text{ALG}(G, \mathcal{O}_{G'}))]$ comes from the effort required to *invalidate* possible s-t paths.

In the first case of ALG (a path exists in both ab and at, or in neither one), it queries more edges than LAZYSP. However, we argue that ALG uses much fewer queries in its second case. The probability of existence of a path in structure B is chosen so that the second case happens with significant probability (almost $\frac{1}{2}$), in which case the overall savings in the analysis of the second case bring down the total expected cost of ALG compared to LAZYSP.

In slightly more detail, the crux of the proof is the case where sb does not have a path and at has a path (an example of the second case of ALG). To invalidate all possible s-t paths, it suffices to certify that structure A on sa does not have a path. Therefore, ALG terminates after querying only one A structure, with high probability, in addition to querying two B structures on sb and at. On the other hand, LAZYSP does not know which one of sb or at has a path, so with probability at least $\frac{1}{2}$ it first queries some A structure other than sa, in which case it has to also query and verify that no path exists in sa. Therefore, LAZYSP has to query 1.5 A structures in expectation. We design structures A and B so that half the cost of checking an additional A structure is much larger than the initial cost that ALG invests in querying edges in two B structures.

The next theorem formalizes the foregoing discussion.

Theorem 2. *There is a graph $G = (V, E)$ and $p \in (0, 1)$ for which the optimal query policy is not in $\mathcal{LazySP}$.*

4.2 Query Complexity Bounds

The previous section implies that algorithms in $\mathcal{LazySP}$ may be suboptimal in the probabilistic setting. Nevertheless, it may still be possible to give satisfying worst-case guarantees with respect to the performance of algorithms in this class. This is exactly what we do next.

Specifically, we show that any algorithm in $\mathcal{LazySP}$ (with an edge selector satisfying a certain property) uses $O(n/p)$ queries, where $n = |V|$, w.h.p. We then show that there is a graph where no correct path-finding algorithm terminates within $\omega(n/p)$ queries. Taken together, these results show that no other algorithm can hope to do significantly better than algorithms in $\mathcal{LazySP}$ over all underlying graphs.

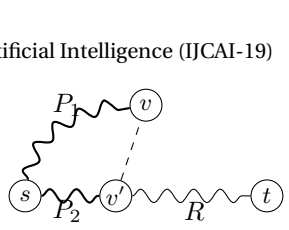

Figure 5: LAZYSP with the forward edge selector does not query an edge between two vertices in the same connected component.

In our upper bound, we focus on edge selectors that choose an unqueried edge between two connected components formed by the validated queried edges.

Definition 1. An edge selector $f : \mathcal{P} \times 2^E \times 2^E$ is *connective* if for any $P \in \mathcal{P}$ and edge sets Q_y and Q_n, $f(P, Q_y, Q_n)$ returns an edge $e \in P \setminus (Q_y \cup Q_n)$ that connects two connected components of the subgraph (V, Q_y).

Let us provide an overview of why the forward edge selector (Section 2) used with a LAZYSP algorithm that breaks ties in favor of paths with more verified edges is connective. Note that at any time the set of verified edges forms a connected component around vertex s. Moreover, by the same reasoning behind Dijkstra [1959], if a vertex v is in that connected component, the shortest s-v path in G' has been found. Now, refer to Figure 5, and consider the path P_1, v, v', R, for two vertices v and v' that are already reachable from s (i.e., $P_1 \subseteq Q_y$ and $P_2 \subseteq Q_y$), and $R \subseteq E \setminus Q$. Then LAZYSP would prefer the path P_2, R, because $|P_2| \leq |P_1| + 1$ (as it is the shortest path to v'), and P_2 is fully verified. We conclude that LAZYSP with the forward edge selector never queries an edge within a connected component.

We now turn to deriving a rigorous upper bound on the number of edges queried by any LAZYSP algorithm with a connective edge selector. In terms of implications, we view this theorem as our main result.

Theorem 3. *For any $\delta > 0$, $p \in (0, 1)$, graph G with n vertices, and a connective edge selector f, with probability at least $1 - \delta$,*

$$\text{cost}(\text{LAZYSP}_f(G, \mathcal{O}_{G'})) \in O\left(\frac{n + \ln(1/\delta)}{p}\right).$$

In the next theorem, we provide a matching lower bound for the number of queries that *any correct path finding algorithm* requires.

Theorem 4. *For all $p \in (0, 1)$ and $n > 15$, there exists a graph G with n vertices such that for any correct path-finding algorithm* ALG*,*

$$\Pr_{G'}\left[\text{cost}(\text{ALG}(G, \mathcal{O}_{G'})) \leq \frac{n-1}{2p}\right] \leq 0.1.$$

References

[Bohlin and Kavraki, 2000] Robert Bohlin and Lydia E. Kavraki. Path planning using lazy PRM. In *IEEE Int.*

Conf. on Robotics and Automation (ICRA), pages 521–528, 2000.

[Choset *et al.*, 2005] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, June 2005.

[Choudhury *et al.*, 2016] Shushman Choudhury, Christopher M. Dellin, and Siddhartha S. Srinivasa. Pareto-optimal search over configuration space beliefs for anytime motion planning. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3742–3749, 2016.

[Choudhury *et al.*, 2017] Shushman Choudhury, Oren Salzman, Sanjiban Choudhury, and Siddhartha S. Srinivasa. Densification strategies for anytime motion planning over large dense roadmaps. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3770–3777, 2017.

[Dellin and Srinivasa, 2016] Christopher M. Dellin and Siddhartha S. Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 459–467, 2016.

[Dijkstra, 1959] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[Haghtalab *et al.*, 2018] Nika Haghtalab, Simon Mackenzie, Ariel D. Procaccia, Oren Salzman, and Siddhartha S. Srinivasa. The provable virtue of laziness in motion planning. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 106–113, 2018.

[Halperin *et al.*, 2017] Dan Halperin, Oren Salzman, and Micha Sharir. Algorithmic motion planning. In *Handbook of Discrete and Computational Geometry, Third Edition*, pages 1311–1342. CRC Press, 2017.

[Hart *et al.*, 1968] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.

[Hauser, 2015] Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2951–2957, 2015.

[Hopcroft *et al.*, 1984] John E Hopcroft, Jacob Theodore Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "warehouseman's problem". *I. J. Robotics Res.*, 3(4):76–88, 1984.

[Hsu *et al.*, 1999] David Hsu, Jean-Claude Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. *Int. J. Comput. Geometry Appl.*, 9(4–5):495–512, 1999.

[Karaman and Frazzoli, 2011] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *I. J. Robotics Res.*, 30(7):846–894, 2011.

[Kavraki *et al.*, 1996] Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics and Automation*, 12(4):566–580, 1996.

[LaValle and Kuffner, 1999] Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 473–479, 1999.

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

[Lozano-Perez, 1983] Tomas Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[Mandalika *et al.*, 2018] Aditiya Mandalika, Oren Salzman, and Siddhartha S. Srinivasa. Efficient shortest-path algorithm for graphs with expensive edge evaluation via lazy lookahead. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 476–484, 2018.

[Mandalika *et al.*, 2019] Aditiya Mandalika, Sanjiban Choushury, Oren Salzman, and Siddhartha S. Srinivasa. Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2019. to appear.

[Salzman and Halperin, 2015] Oren Salzman and Dan Halperin. Asymptotically-optimal motion planning using lower bounds on cost. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 4167–4172, 2015.