The Provable Virtue of Laziness in Motion Planning

Nika Haghtalab  
Computer Science Department  
Carnegie Mellon University

Simon Mackenzie  
Computer Science Department  
Carnegie Mellon University

Ariel D. Procaccia  
Computer Science Department  
Carnegie Mellon University

Oren Salzman  
Robotics Institute  
Carnegie Mellon University

Siddhartha S. Srinivasa  
School of Computer Science & Engineering  
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; 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, Salzman, and Sharir 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, Schwartz, and Sharir 1984), a common approach is to use sampling-based algorithms (Kavraki et al. 1996; Hsu, Latombe, and Motwani 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, Nilsson, and Raphael 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, Salzman, and Srinivasa 2018).

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 extensively evaluate the algorithm empirically on a wide range of edge selectors. Their experiments range from toy scenarios, which demonstrate the advantages of each edge selector, to articulated 7D motion-planning problems that show that, using this approach, non-trivial 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 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; it is explored in Section 4. Here, each edge is endowed with a probability of being in collision — a common assumption in motion planning (see, e.g., Choudhury, Dellin, and Srinivasa 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 connectility. We complement this result with an asymptotic lower bound that holds for any algorithm in the class LazySP, 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 \( \text{ALG} \) 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 \( f : \mathcal{P} \times 2^E \times 2^E \rightarrow E \) 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 \( e \in P \setminus Q \). Examples of edge selectors include:

- **Forward edge selector:** Returns the first unqueried edge in \( P \), that is, the one closest to \( s \).
- **Backward edge selector:** Returns the last unqueried edge in \( P \), that is, the one closest to \( t \).
- **Bisection edge selector:** Returns an unqueried edge in \( P \) which is furthest from an evaluated edge on the path.

Given an edge selector \( f \), the corresponding LazySP is described in Algorithm 1. At a high level, LazySP, 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 \( G' \).

It is not hard to see that any algorithm in the class LazySP is a correct algorithm. This is due to the fact that these algorithms always consider the shortest path that has not yet been ruled out. Therefore, upon termination, they return the shortest \( s \)-\( t \)-path in \( G' \). Moreover, an edge selector never returns an edge that has been queried before and, hence, these algorithms never query an edge more than once. It follows that any such algorithm eventually terminates. See the paper.
of Dellin and Srinivas (2016) for a more detailed discussion of the LazySP class.

Let us conclude this section with an example of the execution of LazySP with the forward edge selector, which also 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 \( sabdet. \)

![Figure 1: Example of the execution of LazySP with the forward edge selector. Solid edges are in \( E' \), dashed edges are in \( E \setminus E' \).](image)

### 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 cost(\( \text{ALG}(G, O_{G'}) \)) denotes the number of edge queries \( \text{ALG} \) 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 \( \text{ALG} \), there exists \( \text{ALG}' \in \text{LazySP} \) such that

\[
\text{cost}(\text{ALG}'(G, O_{G'})) \leq \text{cost}(\text{ALG}(G, O_{G'})).
\]

**Proof.** Let \( P^* \) be a shortest \( s-t \) path in \( G' \) and let \( P_1, \ldots, P_m \) be the \( s-t \) paths in \( G \) that are strictly shorter than \( P^* \), ordered by their length. If no path in \( G' \) exists, \( P_1, \ldots, P_m \) is the list of all \( s-t \) paths in \( G \). Let \( Q \) be the set of edges queried by \( \text{ALG}(G, O_{G'}) \), \( Q_y = Q \cap E' \), and \( Q_n = Q \setminus E' \).

Let \( Q^* \) be the set that includes all edges of \( P^* \), all of which exist in \( G' \), and an optimal cover for the sets \( P_i \) using edges that do not belong to \( G' \). That is, let \( Q^* = P^* \cup Q_n^* \), where

\[
Q_n^* = \arg \min_{S \subseteq E \setminus E'} \{ \sum_{i = 1}^m |P_i \cap S| \}.
\]

We argue that \( |Q| \leq |Q^*| \). This is due to the fact that correctness of \( \text{ALG} \) implies that \( P^* \subseteq Q_y \), and any path that is shorter than \( P^* \) has an invalidated edge, i.e., for all \( i \in [m] \), \( P_i \cap Q_n \neq \emptyset \). Note that the latter condition shows that \( Q_n \) is a cover for the sets \( P_i \) using edges \( E \setminus E' \), so by the optimality of \( Q_n^* \), we have

\[
|Q| = |Q_y| + |Q_n| \geq |P^*| + |Q_n^*| = |Q^*|.
\]

It remains to show that there is \( \text{ALG}' \in \text{LazySP} \) that only queries edges in \( Q^*. \) Let \( \text{ALG}' \) be the algorithm that first queries an edge in \( Q_n^* \), then an edge in \( Q^* \setminus P_k \) (if it is nonempty), and so on, until \( Q_n^* \cap P_m \) (if it is nonempty), and finally queries all the edges in \( P^* \). We argue that \( \text{ALG}' \) must query all the edges in \( Q^* \). Indeed, the only difficulty is that, in principle, it may be the case that at some point \( P_1, \ldots, P_k \) have already been invalidated, and there is some \( e \in Q_n^* \) such that \( e \notin P_1 \cup \cdots \cup P_k \), meaning that \( e \) cannot be queried in the future. But, in that case, \( e \) is not needed in order to invalidate the paths \( P_1, \ldots, P_k \), in contradiction to the optimality of \( Q^* \) (and that of \( Q_n^* \), specifically).

Note that \( \text{ALG}' \) has the property that at any time it only queries edges on the shortest \( s-t \) path that has not been invalidated yet. Clearly, it is possible to define an edge selector that makes the same choices as \( \text{ALG}' \). We conclude that \( \text{ALG}' \), whose cost is at most that of \( \text{ALG} \), can be represented as a member of LazySP.

We can alternatively interpret Theorem 1 in a model where \( \text{LazySP} \) 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

In this section, we consider a probabilistic variant of the setting we investigated in Section 3. We view the probabilistic
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 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”.

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

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 $\ell$ 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. 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 $E[\text{cost}(\text{ALG}(G, \mathcal{O}_{G'}))]$ 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 exam-
ple 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, LAZYS$P$ 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, LAZYS$P$ 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 and its proof formalize 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 Lazy$S$P.

**Proof.** Consider the graph in Figure 2. Let

\[
\kappa = 10^4, \\
\kappa' = 2, \\
\epsilon = 10^{-2}, \\
\delta = 10^{-3}, \\
\ell = \frac{1}{\epsilon} \ln \left( \frac{\kappa}{\delta} \right), \\
\ell' = 3\ell
\]

for the structures in Figures 3 and 4. Let $p = 1 - \epsilon$ be the probability of existence of any one edge in these structures.

In the following claim, we show that the structure in Figure 3 almost certainly does not have a path, but one has to query many edges to verify that this is indeed the case.

**Claim 1.** With probability at least $1 - \delta$, there is no $c$-$d$ path in the structure shown in Figure 3. Conditioned on the event that no $c$-$d$ path exists, any correct querying policy has to query at least $10^6 - 2$ edges in expectation to certify that no path exists.

**Proof.** The probability of a path existing in this structure is exactly

\[
(1 - \epsilon^{\kappa'})^{\ell'} = (1 - 0.01)^2 \frac{3}{2} \ln(10^7) = 0.616 \pm 10^{-3}.
\]

Moreover, since the structure has $\kappa'\ell'$ edges overall, the expected number of queries is also bounded by $\kappa'\ell' \leq 10^4$.

We now turn to comparing the performance of ALG with that of LAZYS$P$. First, note that ALG queries at most $2 \cdot 10^4$ edges for verifying arcs $sb$ and $at$ at the beginning, whereas LAZYS$P$ may not query those edges. Consider the following cases:

1. There is a path in at least one of the $A$ structures $sa$, $ab$, or $bt$.
2. There is no path in the $A$ structures $sa$, $ab$, and $bt$, and exactly one of the $B$ structures $sb$ or $at$ has a path.
3. Cases 1 and 2 do not hold.

Consider Case 1. By Claim 1, this is a rare event that happens with probability at most $3\delta$. Conditioned on this event, ALG verifies at most three $A$ structures in addition to arcs $sb$ and $at$, with overall number of edges $3\kappa\ell$. Taking the probability of this event into account, ALG issues at most

\[3\delta \cdot 3\kappa\ell \leq 1.46 \cdot 10^5\]

more queries in expectation (in addition to the $B$ structures which we will account for separately).

Consider Case 2. By Claims 1 and 2, this event happens with probability at least

\[2(1 - 3\delta) \cdot (0.616 \pm 10^3) \cdot (1 - 0.616 \mp 10^3) \geq 0.471.
\]

Conditioned on this event, ALG invalidates one $A$ structure in addition to verifying arcs $sb$ and $at$. This is due to the fact that ALG only needs to query and invalidate the $A$ structure that is parallel to the non-valid $B$ structure on $sb$ or $at$. For example, when arc $at$ has a path and $sb$ does not, it suffices to invalidate structure $sa$ to certify that no $s$-$t$ path exists in Figure 2.

\[
= 1 - (1 - \epsilon)^\ell - \ell(1 - \epsilon)^\ell \epsilon \\
\geq 1 - 2 \cdot 10^{-4}.
\]

Using the linearity of expectation and summing over all $\kappa$ disjoint strings that have to be invalidated, the expected number of queries needed to invalidate the structure is at least

\[
\kappa \left( 1 - 2 \cdot 10^{-4} \right) = 10^6 - 2.
\]

In the next claim, we show that the structure in Figure 4, though narrower and longer than the structure in Figure 3, has a path with higher probability.

**Claim 2.** With probability $0.616 \pm 10^{-3}$ there is a path in the structure shown in Figure 4. Moreover, the expected number of queries needed to find a path or certify that none exists is at most $10^4$.

**Proof.** The probability of a path existing in this structure is exactly

\[
(1 - \epsilon^{\kappa'})^{\ell'} = (1 - 0.01)^2 \frac{3}{2} \ln(10^7) = 0.616 \pm 10^{-3}.
\]

Moreover, since the structure has $\kappa'\ell'$ edges overall, the expected number of queries is also bounded by $\kappa'\ell' \leq 10^4$.
On the other hand, conditioned on the event that exactly one of the structures $sb$ or $at$ has a path, LAZYSP has to invalidate 1.5 $A$ structures in expectation. Indeed, initially it must query edges on the shortest path, and they are all in $A$ structures. It can only query $sb$ or $at$ after an $A$ structure has been invalidated, but, at that point, with probability 1/2 there might still be a path using another $A$ structure, chained with the valid $B$ structure.

In Case 3, ALG verifies at most 2 more $B$ structures than LAZYSP (this is Case 1 of ALG).

To summarize,
\[
\begin{align*}
\mathbb{E}_{G' \sim G}[\text{cost}(\text{ALG}(G, O_{G'}))] &\leq \mathbb{E}_{G' \sim G}[\text{cost}(\text{AZYSP}(G, O_{G'}))] \\
&+ 2 \cdot 10^4 + 1.46 \cdot 10^5 - 2.35 \cdot 10^5 \quad \text{Case 1}
\end{align*}
\]
\[
\begin{align*}
&< \mathbb{E}_{G' \sim G}[\text{cost}(\text{AZYSP}(G, O_{G'}))]
\end{align*}
\]

It may be instructive to understand why the Example of Figure 2 does not contradict Theorem 1. Consider the potentially problematic Case 2 of the proof of Theorem 2, where, say, the arc $sb$ is collision-free, and the arc $at$ is not; moreover, the three $A$ structures are in collision. Then $Q^* = Q^x = Q^s$, (as defined in the proof of Theorem 1) would be a set of edges that invalidates $at$ and $bt$. In the deterministic setting, LAZYSP with an omniscient edge selector could start by invalidating $bt$, then proceed to $at$.

### 4.2 Query Complexity Bounds

Theorem 2 implies that algorithms in LazySP may be sub-optimal 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 first show that any algorithm in LazySP with an edge selector satisfying a certain property uses $O(n/p)$ queries, where $n = |V|$, with high probability. 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 LazySP over all underlying graphs.

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 $f : P \times 2^E \times 2^E$ is connective if for any $P \in T$ 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 $(V,Q_y)$.

It is not hard to see that the bisection edge selector (defined in Section 2) is not a connective edge selector. On the other hand, both forward and backward edge selectors are connective.

Let us provide an overview of why the forward edge selector — used with a LAZYSP algorithm that breaks ties in favor of paths with more verified edges — is connective (the same argument applies to the backward edge selector, switching the roles of $s$ and $t$). 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. Although the proof seems straightforward in retrospect, it is actually rather tricky. 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 $1 - \delta$,
\[
\text{cost}(\text{AZYSP}_f(G, O_{G'})) = O \left( \frac{n + \ln(1/\delta)}{p} \right).
\]

**Proof.** Let
\[
m = \frac{1}{p} \max \left\{ 2n, 8 \ln \left( \frac{1}{\delta} \right) \right\},
\]
and let $E'$ be the event that $\text{cost}(\text{AZYSP}_f(G, O_{G'})) > m$.

Consider $m$ independent Bernoulli random variables each with parameter $p$, $X = (X_1, \ldots, X_m)$. Let $X_i = 1$ correspond to the event where the $i$th edge queried by LazySP is in $E'$, and $X_i = 0$ otherwise. Intuitively, we think of $X_1, \ldots, X_m$ as flipping coins with bias $p$ in advance to decide the answers to the queries issued by LazySP. We can do this because the probability that the answer to a query is YES is independent of which edge is queried.

Formally, let $P_{(X,G',s,G')}$ correspond to taking probability over a random process that generates $G'$ by first instantiating the Bernoulli random variables $X_1, \ldots, X_m$, then determining the corresponding sets of edges $Q_y$ and $Q_n$, that are validated and invalidated by LazySP, respectively. For any edge $e \in Q_y$ or $e \in Q_n$, set $e$ to belong to, or not belong to $E'$, respectively. For any edge $e \in E \setminus (Q_y \cup Q_n)$, set $e$
to belong to $E'$ with probability $p$, independently. Note that in this process each edge belongs to $E'$ with probability exactly $p$. So, $\Pr_{G\rightarrow G'}[\mathcal{E}] = \Pr((X,G'),(X')^c)$. Using conditional probability, we have

$$\Pr_{G'}[\mathcal{E}] = \Pr_{(X,G')} \left[ \sum_{i=1}^{m} X_i < n \right] \Pr_{(X,G')} \left[ \sum_{i=1}^{m} X_i < n \right] \leq \Pr_{X'} \left[ \sum_{i=1}^{m} X_i < n \right] \leq \Pr_{X'} \left[ \sum_{i=1}^{m} X_i < n \right] \leq \delta,$$

where the last inequality is a direct consequence of the Chernoff bound:

$$\Pr_{X'} \left[ \sum_{i=1}^{m} X_i < n \right] \leq \exp \left( -\frac{mp}{8} \right).$$

Next, we argue that the second term, in Equation (2), is zero. Specifically, we show that

$$\Pr_{(X,G')} \left[ \sum_{i=1}^{m} X_i \geq n \right] = 0.$$  (3)

Indeed recall that $Q_o$ denotes the set of edges validated by $\text{LAZYSP}_f$ during the first $m$ queries, corresponding to $X_1, \ldots, X_m$. Note that if $\text{LAZYSP}_f$ does not terminate within the first $m$ queries, then $s$ and $t$ belong to two different connected components of the graph $H = (V,Q_o)$. By the connectivity property of $f$, at any time $\text{LAZYSP}_f$ only queries an edge that is between two connected components of the validated edges at that time. So, every time $\text{LAZYSP}_f$ encounters a queried edge that is realized (that is, $Q_o$ grows), the number of connected components in $H$ decreases. Therefore, after encountering $n$ verified edges, i.e., $\sum_{i=1}^{n} X_i \geq n$, there is an $s$-$t$ path in $H$. This establishes Equation (3).

Putting terms (1) and (2) together, we have that $\Pr_{G'}[\mathcal{E}] \leq \delta$.  

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 $\text{ALG}$,

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

**Proof.** Let $m = \lfloor (n-1)/2p \rfloor$. Consider the following graph $G = (V,E)$: Let $V$ be the sequence of vertices $s = v_1, v_2, \ldots, v_n = t$. For each $i = 1, \ldots, n-1$, let there be $m+1$ parallel edges between $v_i$ and $v_{i+1}$.

Since there are $m+1$ parallel edges between any two vertices, $\text{ALG}$ cannot certify that no $s$-$t$ path exists in $G'$ with only $m$ queries. So, if $\text{ALG}$ terminates with at most $m$ queries, it is because it has found an $s$-$t$ path. To find an $s$-$t$ path, $\text{ALG}$ must have encountered at least $n - 1$ realized edges between the $m$ queries it has made. Therefore, using the same Bernoulli random variables as in the proof of Theorem 3, the Chernoff bound, and the fact that $n > 15$, we have

$$\Pr_{G'} \left[ \text{cost}(\text{ALG}(G,O_{G'})) \leq m \right] \leq \Pr_{X_1,\ldots,X_m} \left[ \sum_{i=1}^{m} X_i \geq n - 1 \right] \leq \Pr_{X_1,\ldots,X_m} \left[ \sum_{i=1}^{m} X_i \geq 2mp \right] \leq \exp(-mp/3) \leq 0.1.$$

**5 Discussion**

We wrap up by briefly discussing some pertinent issues.

**Multigraphs are mostly for ease of exposition.** Recall that the graph $G$ can have multiple edges between two vertices. As we mentioned in Section 2, this assumption is “mostly” for ease of exposition. Clearly, our positive results, Theorems 1 and 3, hold even for simple graphs. Our first negative result, Theorem 2, holds for simple graphs but the construction becomes (even) more unwieldy. The only exception is Theorem 4: we were unable to establish it for simple graphs (it is easy to prove a lower bound of $\Omega(n/n\ln n)$ for constant $p$, though).

**Is the connectivity assumption needed?** Theorem 3 holds for $\text{LAZYSP}$ with a connective edge selector. Even though the current proof strongly relies on the connectivity assumption, we have not found examples of edge selectors that violate the theorem’s conclusion (in particular, we have not been able to find a bad example for the bisection edge selector). We therefore conjecture that the $O(n/p)$ query complexity upper bound holds for any edge selector, as long as $\text{LAZYSP}$ breaks ties in favor of paths with more verified edges (otherwise it is easy to construct bad examples). Despite significant effort on our part, this conjecture remains open.

**Should we consider edge evaluations or shortest paths?** A natural concern that could be raised is that this paper considers only shortest paths in terms of number of edges traversed, instead of shortest paths in a directed graph. Is the unweighted graph case that was analyzed in this paper sufficient to deal with the applications mentioned in the introduction? The answer is yes. Indeed, evaluating whether an edge in a motion-planning roadmap is collision free or not is typically done by finely discretizing the edge. Every discretized point is associated with a robot configuration, which can be tested for collision against all physical obstacles. Thus, the computational cost of testing if an edge is collision free or not is proportional to its length. Importantly, if we take a directed graph (which is a motion-planning roadmap) and subdivide each edge into short edges of constant length (except,
maybe, the last edge), we obtain an unweighted graph. Minimizing the number of edge evaluations in this unweighted graph is roughly proportional to minimizing the number of collision checks for robot configurations in the original directed graph.

Computation of the optimal policy in the probabilistic setting. Our results suggest that LAZYSP is an excellent proxy for the optimal policy in the probabilistic setting, in that with, say, the forward edge selector, it is computationally efficient and provides satisfying guarantees. This is backed up by the empirical evaluation presented by Dellin and Srinivasa (2016). One may ask, though, whether the optimal policy itself can be computed. The answer is that this seems to be an extremely hard problem. The most direct representation of the problem is via a Markov decision process (MDP), where there is a state for every possible choice of $Q_y$ and $Q_n$, the action space is edges in $E \setminus Q$, and the transitions and rewards are defined in the obvious way. Although an optimal policy in an MDP can be computed in polynomial time in its representation, the difficulty is that the size of the state space is exponential in $|E|$. That said, heuristics for (exactly or approximately) computing the optimal policy in the probabilistic setting have the potential to provide a practical alternative to LAZYSP.

6 Acknowledgments

This work was partially supported by NSF grants IIS-1409003, IIS-1350598, IIS-1714140, CCF-1525932, and CCF-1733556; by ONR grants N00014-16-1-3075 and N00014-17-1-2428; and by a Sloan Research Fellowship, an MSR PhD Fellowship and a Siebel Scholarship.

References


