Generalized Lazy Search for Robot Motion Planning: Interleaving Search and Edge Evaluation via Event-based Toggles

Aditya Mandalika
University of Washington
adityavk@cs.uw.edu [*]

Sanjiban Choudhury
University of Washington
sanjibac@cs.uw.edu [*]

Oren Salzman
Carnegie Mellon University
osalzman@andrew.cmu.edu [*]

Siddhartha Srinivasa
University of Washington
siddh@cs.uw.edu [*]

Abstract

Lazy search algorithms can efficiently solve problems where edge evaluation is the bottleneck in computation, as is the case for robotic motion planning. The optimal algorithm in this class, LazySP, lazily restricts edge evaluation to only the shortest path. Doing so comes at the expense of search effort, i.e., LazySP must *recompute the search tree* every time an edge is found to be invalid. This becomes prohibitively expensive when dealing with large graphs or highly cluttered environments. Our key insight is the need to balance both edge evaluation and search effort to minimize the total planning time. Our contribution is two-fold. First, we propose a framework, Generalized Lazy Search (GLS), that seamlessly toggles between search and evaluation to prevent wasted efforts. We show that for a choice of toggle, GLS is provably more efficient than LazySP. Second, we leverage prior experience of edge probabilities to derive GLS policies that minimize expected planning time. We show that GLS equipped with such priors significantly outperforms competitive baselines for many simulated environments in $\mathbb{R}^2$, $SE(2)$ and 7-DoF manipulation.

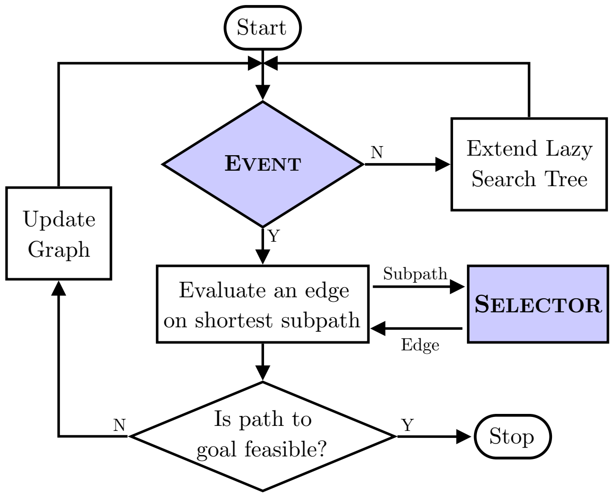

Figure 1: The Generalized Lazy Search (GLS) framework with two parameters - Event and Selector (blue)

1 Introduction

We focus on the problem of finding the shortest path on a graph while minimizing total planning time. This is critical in applications such as robotic motion planning (LaValle 2006), where collision-free paths must be computed in real time. A typical search algorithm expands a wavefront from the start, evaluating edges discovered until it finds the shortest feasible path to the goal. The planning time then becomes the sum of the time spent in two phases – *search effort* and *edge evaluation*. While edge evaluation is generally more expensive in motion planning (Hauser 2015), the *actual ratio of these times varies* with problem instances and graph sizes. Our goal is to design a framework of algorithms that let us balance this trade-off.

Unfortunately, current shortest path algorithms do not provide a framework flexible enough to traverse the pareto curve between search effort and edge evaluation. On one end of the spectrum, A* and its variants (Hart, Nilsson, and Raphael 1968; Yoshizumi, Miura, and Ishida 2000; Korf 1985) evaluate edges as soon as they are discovered. Hence although A* is optimal in terms of *search effort*, it is at the cost of excessive edge evaluations. On the other hand, LazySP (Dellin and Srinivasa 2016) amongst other lazy search techniques (Bohlin and Kavraki 2000; Cohen, Phillips, and Likhachev 2014; Hauser 2015), expands the search wavefront all the way to the goal before evaluating edges. Hence LazySP is optimal in terms of edge evaluation but has to replan everytime an edge is invalidated.

In this work, we propose a framework for *algorithmically toggling* between search effort and edge evaluation. We are guaranteed to find the shortest path as long as the following holds true; the search tree must always be repaired to be consistent, and edge evaluation must be restricted to the shortest subpath in the tree. Our framework, *Generalized Lazy Search* (GLS), has two modules - Event and Selector (Fig. 1). The algorithm expands a lazy search tree without evaluating any edges till the Event is triggered. A Selector is then invoked to evaluate an edge on the shortest subpath in the lazily expanded search tree. We show that by choosing

[*]This work was (partially) funded by the National Institute of Health R01 (#R01EB019335), National Science Foundation CPS (#1544797), National Science Foundation NRI (#1637748), the Office of Naval Research, the RCTA, Amazon, and Honda.

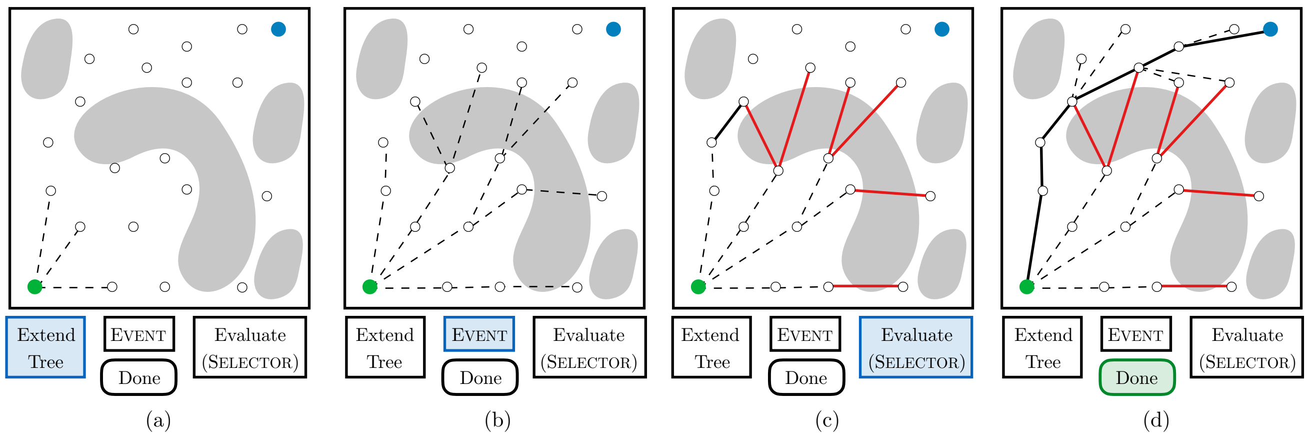

Figure 2: Mechanics of the GLS framework (Algorithm 1) for an ideal Event and Selector combination.

different Event and Selector pairs, we can recover several existing lazy search algorithms such as LazySP (Dellin and Srinivasa 2016), LWA* (Cohen, Phillips, and Likhachev 2014) and LRA* (Mandalika, Salzman, and Srinivasa 2018).

What constitutes an optimal trade-off and can this be captured by GLS? Consider the ideal scenario, one with an omniscient oracle (Haghtalab et al. 2018) that knows ahead of time which edges are valid or invalid. In fact, the oracle can compute the minimal set of invalid edges $\mathcal{I}$ that must be invalidated to arrive at the shortest feasible path. How can we utilize such an oracle in GLS? A simple strategy is as follows; as the search wavefront expands from start to goal, the oracle monitors the new edges that are discovered and triggers an Event if it belongs to $\mathcal{I}$. A Selector then evaluates that edge. This minimizes edge evaluation and curtails wasted search effort.

This insight extends to the more practical setting where we have *priors on edge validity* that are learned from experience. We derive Event and Selector that minimize the expected planning time. This produces behaviors similar to the omniscient oracle (Fig. 2); the search proceeds until the Event is triggered due to the appearance of low probability edges on the current subpath; the Selector then selects these edges to invalidate the subpath; and the process continues until the shortest feasible path is found.

We make the following contributions:

1. We propose a class of algorithms, GLS (Section 4), that minimize computational effort, defined as a function of both edge evaluation and vertex rewiring (Section 3).
2. We recover different lazy search algorithms as instantiations of GLS. We further prove that one such instantiation is edge optimal and causes fewer rewires than LazySP (Section 4, Theorem 4.3).
3. We derive instantiations of GLS that exploit the availability of edge priors to minimize expected computational effort (Section 5, Theorem 5.2).
4. We show that GLS informed with edge priors can outperform competitive baselines on a spectrum of planning domains (Section 6).

2 Related Work

Graphs lend powerful tractability to robotic motion planning (LaValle 2006). They can be explicit, i.e., constructed as part of a pre-processing stage (Kavraki et al. 1996; Karaman and Frazzoli 2011; Janson et al. 2015), or implicit, i.e., discovered incrementally during search (Likhachev, Gordon, and Thrun 2004; Gammell, Srinivasa, and Barfoot 2015; Salzman and Halperin 2015).

A* (Hart, Nilsson, and Raphael 1968) and its variants have enjoyed widespread success in finding the shortest path with an optimal number of vertex expansions. However, in domains where edge evaluations are expensive and dominate the planning time, a *lazy approach* is often employed (Bohlin and Kavraki 2000; Hauser 2015; Kim, Kwon, and Yoon 2018). In this approach, the graph is constructed *without* testing if edges are collision-free. Only a subset of edges are evaluated to save computation time. LazySP (Dellin and Srinivasa 2016) extends the graph up to the goal before checking edges. LWA* (Cohen, Phillips, and Likhachev 2014) extends the graph a single step before evaluation. LRA* (Mandalika, Salzman, and Srinivasa 2018) trades off these approaches, allowing the search to proceed to an arbitrary lookahead. We generalize this further by introducing an event-based toggle.

Several works have explored the use of priors in search. FuzzyPRM (Nielsen and Kavraki 2000) evaluates paths that minimize the probability of collision. The Anytime Edge Evaluation (AEE*) framework (Narayanan and Likhachev 2017) uses an anytime strategy for edge evaluation informed by priors. POMP (Choudhury, Dellin, and Srinivasa 2016) defines surrogate objectives using priors to improve anytime planning. BISECT (Choudhury et al. 2017) and DIRECT (Choudhury, Srinivasa, and Scherer 2018) cast search as Bayesian active learning to derive edge evaluation. E-graphs (Phillips et al. 2012) uses priors in heuristics. We focus on using priors to find the shortest path while minimizing expected planning time.

Several alternate approaches speed up planning by creating efficient data structures (Bialkowski et al. 2016), modeling belief over the configuration space (Huh and Lee 2016), sampling vertices in promising regions (Bialkowski, Otte, and Frazzoli 2013; Burns and Brock 2005) or using specialized hardware (Murray et al. 2016). Other approaches forego optimality and computing near-optimal paths (Salzman and Halperin 2016; Dobson and Bekris 2014). Our work also draws inspiration from approaches that interleave planning and execution, such as LRTA* (Korf 1990) and LSS-LRTA* (Koenig and Sun 2009).

3 Problem Formulation

Our goal is to design an algorithm that can solve the Single Source Shortest Path (SSSP) problem while minimizing computational effort. We begin with the SSSP problem. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V}$ denotes the set of vertices and $\mathcal{E}$ the set of edges. Given a pair of source and target vertices $(v_{\mathrm{s}}, v_{\mathrm{t}}) \in \mathcal{V}$, a path σ is represented as a sequence of vertices $(v_1, v_2, \dots, v_l)$ such that $v_1 = v_{\mathrm{s}}, v_l = v_{\mathrm{t}}, \forall i,\ (v_i, v_{i+1}) \in \mathcal{E}$. We define a *world* $\phi : \mathcal{E} \to \{0, 1\}$ as a mapping from edges to valid (1) or invalid (0). A path is said to be *feasible* if all edges are valid, i.e., $\forall e \in \sigma, \phi(e) = 1$. Let $w : \mathcal{E} \to \mathbb{R}^+$ be the length of an edge. The length of a path is the sum of edge costs, i.e., $w(\sigma) = \sum_{e \in \sigma} w(e)$. The objective of the SSSP problem is to find the shortest feasible path:

$$\min_{\sigma} \ w(\sigma) \quad \text{s.t.} \quad \forall e \in \sigma, \phi(e) = 1 \tag{1}$$

Given an SSSP, we define a shortest path algorithm $\textsc{Alg}(\mathcal{G}, v_{\mathrm{s}}, v_{\mathrm{t}}, \phi)$ that takes as input the graph $\mathcal{G}$, the source-target pair $(v_{\mathrm{s}}, v_{\mathrm{t}})$, and the underlying world ϕ. The algorithm typically solves the problem by building, verifying and rewiring a shortest path tree from source to target.

Maintaining the search tree and verifying the shortest feasible path are primarily characterized by two atomic operations: edge evaluation and vertex rewiring.

Definition 3.1 (Edge Evaluation)**.** *The operation of querying the world $\phi(e)$ to check if an edge e is valid.*

Definition 3.2 (Vertex Rewiring)**.** *The operation of finding and assigning a new parent for a vertex u when an invalid edge is discovered.*

The algorithm returns three terms, i.e, $\sigma^*,\ \mathcal{E}_{\mathrm{eval}},\ \mathcal{V}_{\mathrm{rwr}} = \textsc{Alg}(\mathcal{G}, v_{\mathrm{s}}, v_{\mathrm{t}}, \phi)$. Here, σ^* is the shortest feasible path, $\mathcal{E}_{\mathrm{eval}}$ is the *set* of edges evaluated during the search, and $\mathcal{V}_{\mathrm{rwr}}$ is the *multiset*[1] of vertices rewired. Alg ensures the following certificate:

1. Returned path σ^* is verified to be feasible, i.e., $\forall e \in \sigma^*,\ e \in \mathcal{E}_{\mathrm{eval}},\ \phi(e) = 1$
2. All paths shorter than σ^* are verified to be infeasible, i.e., $\forall \sigma_i,\ w(\sigma_i) \leq w(\sigma^*),\ \exists e \in \sigma_i,\ e \in \mathcal{E}_{\mathrm{eval}},\ \phi(e) = 0$

We now define the computational cost (planning time), of solving the SSSP problem as a function of $\mathcal{V}_{\mathrm{rwr}}$ and $\mathcal{E}_{\mathrm{eval}}$. Let c_e be the average cost of evaluating an edge, and c_r be the average cost of rewiring a vertex. We approximate the total planning time as a linear combination:

$$C(\mathcal{E}_{\mathrm{eval}}, \mathcal{V}_{\mathrm{rwr}}) = c_e \left|\mathcal{E}_{\mathrm{eval}}\right| + c_r \left|\mathcal{V}_{\mathrm{rwr}}\right| \tag{2}$$

[1]$\mathcal{V}_{\mathrm{rwr}}$ is a multiset since a vertex can potentially be rewired multiple times during the planning cycle.

Algorithm 1: Generalized Lazy Search

```
Input      : Graph $\mathcal{G}$, source $v_{\mathrm{s}}$, target $v_{\mathrm{t}}$, world $\phi$
Parameter  : Event, Selector
Output     : $\sigma^*$, $\mathcal{E}_{\mathrm{eval}}$, $\mathcal{V}_{\mathrm{rwr}}$

1 $\mathcal{E}_{\mathrm{eval}} \leftarrow \emptyset, \mathcal{V}_{\mathrm{rwr}} \leftarrow \emptyset$
2 $\mathcal{T}_{\mathrm{lazy}} \leftarrow \{v_{\mathrm{s}}\}$                          ▷ Initialize
3 repeat
4     $\mathcal{T}_{\mathrm{lazy}} \leftarrow$ ExtendTree (Event, $\mathcal{T}_{\mathrm{lazy}}$)  ▷ Add $\mathcal{V}_{\mathrm{rwr}}$
5     $\sigma_{\mathrm{sub}} \leftarrow$ GetShortestPathToLeaf ($\mathcal{T}_{\mathrm{lazy}}$)
6     EvaluateEdge (Selector, $\sigma_{\mathrm{sub}}$)       ▷ Add $\mathcal{E}_{\mathrm{eval}}$
7 until shortest feasible path found
    s.t. $\forall e \in \sigma^*, \phi(e) = 1$;
```

Our motivation for defining the cost will become clearer in the following section, where we propose a general framework for Alg. This framework lets us explicitly reason about the terms $\mathcal{E}_{\mathrm{eval}}$ and $\mathcal{V}_{\mathrm{rwr}}$ in order to balance them.

4 Generalized Lazy Search

We propose a framework, Generalized Lazy Search (GLS), to solve the problem defined in Section 3. The general concept idea is to *toggle* between lazily searching to a horizon and evaluating edges along the current estimated shortest path. This toggle must be chosen appropriately to balance the competing computational costs of edge evaluation and vertex rewiring.

4.1 The Algorithm

Algorithm 1 describes the GLS framework for the shortest path algorithm $\textsc{Alg}(\mathcal{G}, v_{\mathrm{s}}, v_{\mathrm{t}}, \phi)$ referred to in Section 3. This framework requires two functions: Event and Selector.

To solve the SSSP problem, we maintain a shortest path search tree over $\mathcal{G}$. We assume that every call to ϕ, which populates $\mathcal{E}_{\mathrm{eval}}$, is expensive. Therefore, we initially assume that all edges in $\mathcal{G}$ are valid and maintain this search tree *lazily*. Our algorithm initializes the search tree $\mathcal{T}_{\mathrm{lazy}}$ rooted at v_{s} (Line 1). It begins by iteratively extending $\mathcal{T}_{\mathrm{lazy}}$ into $\mathcal{G}$ (Line 4). The search is guided with an admissible heuristic $h(v, v_{\mathrm{t}})$.

The procedure ExtendTree additionally takes as input a function Event. Extending $\mathcal{T}_{\mathrm{lazy}}$ triggers the Event by definition. The algorithm, at this point, discontinues the extension of $\mathcal{T}_{\mathrm{lazy}}$ and switches to validate the already constructed search tree. Therefore, the Event acts as a toggle between lazy seach and edge evaluation.

Definition 4.1 (Event)**.** *A function that defines the toggle between extending the lazy search tree and validating it.*

To solve the SSSP problem and validate $\mathcal{T}_{\mathrm{lazy}}$, the algorithm picks the path, σ_{sub}, to a leaf vertex with the lowest estimated total cost to reach the goal (Line 5). It then evaluates an edge along σ_{sub} to validate the search tree (Line 6). In addition to σ_{sub}, the procedure EvaluateEdge also takes as input a function Selector. The Selector acts on σ_{sub} and returns an edge belonging to it that the algorithm evaluates.

Definition 4.2 (SELECTOR). *A function that defines the strategy to select an edge along a subpath to evaluate.*

Edge evaluation is followed by the extension of $\mathcal{T}_{\text{lazy}}$ until the EVENT is triggered again. If the edge were invalid, the subtree emanating from the edge has to be rewired. We can do this efficiently using the mechanics of LPA* (Koenig, Likhachev, and Furcy 2004).

This process of interleaving search with edge evaluation continues until the algorithm terminates with the shortest feasible path from source to goal, if one exists. While the algorithm is guaranteed to return the shortest path, the framework permits the design of EVENT and SELECTOR to reduce the total computation cost of solving the SSSP problem.

Algorithm 2: Candidate EVENT Definitions

```
$v \leftarrow$ leaf vertex in $\mathcal{T}_{\text{lazy}}$ with least estimated cost to $v_t$
Function ShortestPath()
    if $v = v_t$ then
        return true;
    end
Function ConstantDepth(depth $\alpha$)
    $\sigma_{\text{sub}} \leftarrow$ path from $v_s$ to $v$
    $\alpha_v \leftarrow$ number of unevaluated edges in $\sigma_{\text{sub}}$
    if $\alpha_v = \alpha$ or $v = v_t$ then
        return true;
    end
Function HeuristicProgress
    $h_{\min} \leftarrow \min_{(u',v') \in \mathcal{E}_{\text{eval}}} h(v', v_t)$
    if $h(v, v_t) < h_{min}$ or $v = v_t$ then
        return true;
    end
    return;
Function SubpathExistence(probability $\delta$)
    $\sigma_{\text{sub}} \leftarrow$ path from $v_s$ to $v$
    $p \leftarrow \prod_{e \in \sigma} \mathbf{p}(e)$
    if $p \leq \delta$ or $v = v_t$ then
        return true;
    end
```

Algorithm 3: Candidate SELECTOR Definitions

```
Function Forward()
    return {first unevaluated edge closest to $v_s$};
Function Alternate()
    if Iteration Number is Odd then
        return {first unevaluated edge closest to $v_s$};
    end
    else
        return {first unevaluated edge closest to $v_t$};
    end
Function FailFast()
    return $\{\arg\min_{e \in \sigma_{\text{sub}}} \mathbf{p}(e)\}$;
```

4.2 Role of EVENT and SELECTOR

Since the lazy search paradigm operates based on the concept of optimism under uncertainty, the search tree is extended assuming edges are collision free. However, extending the search tree beyond edges that are in collision can waste computational effort. The EVENT acts as a toggle to halt a search deemed wasteful. The SELECTOR aims to quickly invalidate the path. Fig. 2 illustrates the ideal behavior of such an algorithm. Interestingly, the framework can also capture existing lazy search algorithms as different combinations of event and selectors, as shown in Table. 1.

EVENT. When triggered, events must ensure that the shortest subpath σ_{sub} in $\mathcal{T}_{\text{lazy}}$ has at least one unevaluated edge (Theorem 4.1). Algorithm 2 defines some candidate events.

SHORTESTPATH (SP) is triggered when a shortest path to v_t has been determined during the lazy extension of $\mathcal{T}_{\text{lazy}}$. Therefore, in every iteration, this EVENT presents the SELECTOR with the candidate shortest path from v_s to v_t on $\mathcal{G}$. Note that SHORTESTPATH exhibits algorithmic behavior similar to LazySP and LazyPRM.

CONSTANTDEPTH (CD) is triggered when the procedure EXTENDTREE chooses to extend a leaf vertex $v \in \mathcal{T}_{\text{lazy}}$ such that the subpath from $v_s$to v has exactly α number of unevaluated edges. Therefore, in every iteration, this EVENT presents the SELECTOR with σ_{sub} that is characterized by a constant number of unevaluated edges.

HEURISTICPROGRESS (HP) is triggered whenever the search expands a vertex whose heuristic value is lower than any vertex whose incident edge has been evaluated. It does so by recording the minimum heurisitic value of a vertex with a parent that has been evaluated, i.e., $h_{\min} \leftarrow \min_{(u',v') \in \mathcal{E}_{\text{eval}}} h(v', v_t)$. The event is triggered whenever EXTENDTREE chooses to extend a leaf vertex $v \in \mathcal{T}_{\text{lazy}}$ with a heuristic value smaller than $h_{\min}$.

SELECTOR. Selectors must ensure that they select at least one unevaluated edge (Theorem 4.1). Algorithm 3 defines some candidate selectors.

Given σ_{sub}, FORWARD (F) evaluates the first unevaluated edge on σ_{sub} that is closest to v_s. Given a forward search, this constitutes one of the most natural SELECTORS available. ALTERNATE (A) toggles between evaluating the first unevaluated edge closest to v_s and v_t in every iteration. This approach is motivated by bi-directional search algorithms. Both SELECTORS were first used in (Dellin and Srinivasa 2016).

Algorithm	EVENT	SELECTOR
LazyPRM (2000)	SHORTESTPATH	Any
LazySP (2016)	SHORTESTPATH	Any
LWA* (2014)	CONSTANTDEPTH (1)	FORWARD
LRA* (2018)	CONSTANTDEPTH (α)	FORWARD

Table 1: Equivalence of GLS to existing lazy algorithms

4.3 Analysis

For any choice of Event and Selector, GLS is complete and correct.

Theorem 4.1 (Completeness). *Let Event be a function that on halting ensures there is at least one unevaluated edge on the current shortest path or that the goal is reached. Let Selector be a function that evaluates at least one unevaluated edge (if it exists). GLS implemented using ExtendTree(Event) and EvaluateEdges(Selector) on a finite graph $\mathcal{G}$ is complete.*

Proof In each iteration, ExtendTree(Event) ensures there is atleast one unevaluated edge on the shortest path (unless the goal has been reached). The EvaluateEdges(Selector) evaluates atleast one edge. Since there are a finite number of edges, the algorithm will eventually terminate. □

Theorem 4.2 (Correctness). *If the heuristic $h(v, v_\mathrm{t})$ is admissible, then GLS terminates with the shortest feasible path.*

Proof Let σ^* be the shortest feasible path with respect to $w(\cdot)$ and world ϕ. For any vertex $v^* \in \sigma^*$, we denote its f-value to be $f(v^*) = w(\sigma_{v_\mathrm{s},v^*}) + h(v^*, v_\mathrm{t})$, where $\sigma_{v_\mathrm{s},v^*}$ is the subpath from the start to vertex v^*. As our heuristic function is admissible, we have that $f(v^*) \leq w(\sigma^*)$. Recall that in each iteration, the inner GetShortestPathToLeaf () returns a vertex v_ret with the smallest f-value among all the leaves of the tree $\mathcal{T}_\mathrm{lazy}$. Let v^*_leaf be the leaf vertex on $\mathcal{T}_\mathrm{lazy}$ that lies on the shortest feasible path σ^*. Hence, $f(v_\mathrm{ret}) \leq f(v^*_\mathrm{leaf}) \leq w(\sigma^*)$. If GLS terminates with v_ret, this implies $\sigma_{v_\mathrm{s},v_\mathrm{ret}}$ is verified to be feasible and $v_\mathrm{ret} = v_\mathrm{t}$. Let the verified path that is returned be σ_ret such that $\sigma_\mathrm{ret} = \sigma_{v_\mathrm{s},v_\mathrm{t}}$. In that case $w(\sigma_\mathrm{ret}) = f(v_\mathrm{t}) \leq w(\sigma^*)$. □

LazySP with the Forward selector was proved to be *edge optimal*[2] in the class of all shortest path algorithms that use a Forward selector (Mandalika, Salzman, and Srinivasa 2018). We now show that GLS lets us derive another algorithm that is *also edge-optimal* but reduces number of vertex rewires.

Theorem 4.3 (Edge Optimality). *GLS evaluates the same number of edges $\mathcal{E}_\mathrm{eval}$ as LazySP, i.e., is edge optimal, while having a smaller number of vertex rewires $\mathcal{V}_\mathrm{rwr}$ under the following setting:*

1. *Heuristic: Distance on the unevaluated graph $h_\mathcal{G}(v, v_\mathrm{t})$*
2. *Event: HeuristicProgress*
3. *Selector: Forward*

Proof We are going to prove this via induction over iterations of LazySP and GLS. In each iteration cycles through algorithm by invoking EvaluateEdges (Selector), ExtendTree (Event) and SelectShortestSubpath ().

At iteration i, let $\mathcal{E}^i_\mathrm{eval,LSP}$ and $\mathcal{V}^i_\mathrm{rwr,LSP}$ be the edges evaluated and vertex rewired respectively by LazySP. Let σ^i be the candidate shortest path.

Let $\mathcal{E}^i_\mathrm{eval,GLS}$ and $\mathcal{V}^i_\mathrm{rwr,GLS}$ be the edges evaluated and vertices rewired, respectively, by GLS at iteration i. Let $h_\mathcal{G}(v, v_\mathrm{t})$ be the heuristic used by the search which corresponds to the distance on the graph $\mathcal{G}$. Let v^i be the leaf vertex corresponding to the current shortest subpath from the start $\sigma_{v_\mathrm{s},v^i}$. This implies v^i corresponds to the vertex with the smallest f-value $w(\sigma_{v_\mathrm{s},v^i}) + h_\mathcal{G}(v^i, v_\mathrm{t})$.

We also introduce the lazy edge status function $\phi_\mathrm{lazy}(\sigma, \mathcal{E}_\mathrm{eval})$ which determines if a path σ is valid depending on edges evaluated thus far in $\mathcal{E}_\mathrm{eval}$.

Following are the conditions for the induction:

A Both algorithms have the same set of evaluated edges $\mathcal{E}^i_\mathrm{eval,GLS} = \mathcal{E}^i_\mathrm{eval,LSP}$.

B Both algorithms share the same subpath $\sigma^i_{v_\mathrm{s},v} \subseteq \sigma^i$.

For $i = 1$, $\mathcal{E}^1_\mathrm{eval,GLS} = \mathcal{E}^1_\mathrm{eval,LSP}$ because no edges have been evaluated. Hence (A) is true. Since $h_\mathcal{G}(v, v_\mathrm{t})$ is the distance on the unevaluated graph, the leaf vertex v^i considered by GLS lies on σ^1, i.e. $\sigma^1_{v_\mathrm{s},v} \subseteq \sigma^1$. Hence (B) is true.

We will show these conditions hold for $i + 1$.

Since both LazySP and GLS use Forward, share the same subpath (A) and have the same evaluation status (B) - they both evaluate the same edge e. Both algorithms increase their evaluated set $\mathcal{E}^{i+1}_\mathrm{eval,LSP} = \mathcal{E}^{i+1}_\mathrm{eval,GLS} \leftarrow \mathcal{E}^i_\mathrm{eval,GLS} \cup e$. Hence (A) holds.

If e is valid, neither algorithms rewire vertices. However, if an edge is in collision, LazySP rewires at least the remainder of the path σ^{i+1}. GLS does not have to rewire the remainder of the subpath $\sigma_{v^i,v_\mathrm{t}}$ as it was never expanded during the search. Hence GLS can only result in smaller rewires, i.e. $|\mathcal{V}^{i+1}_\mathrm{rwr,GLS}| \leq |\mathcal{V}^{i+1}_\mathrm{rwr,LSP}| - |\sigma_{v^i,v_\mathrm{t}}|$.

We will now show that $\sigma_{v_\mathrm{s},v^{i+1}} \subseteq \sigma^{i+1}$.

LazySP finds the next candidate shortest path σ^{i+1} by solving the following search problem

$$\begin{aligned} \sigma^{i+1} \leftarrow \arg\min_{\sigma} \quad & w(\sigma) \\ \text{s.t.} \quad & \phi_\mathrm{lazy}(\sigma, \mathcal{E}^{i+1}_\mathrm{eval,LSP}) = 1 \end{aligned} \tag{3}$$

GLS invokes the ExtendTree (Event) which proceeds till HeuristicProgress toggles off the search. The search stops at vertex v^{i+1} which satisfies the following:

$$\begin{aligned} v^{i+1} \leftarrow \arg\min_{v} \quad & w(\sigma_{v_\mathrm{s},v}) + h_\mathcal{G}(v, v_\mathrm{t}) \\ \text{s.t.} \quad & h_\mathcal{G}(v, v_\mathrm{t}) < h_\mathrm{min} \end{aligned} \tag{4}$$

Note that $\phi_\mathrm{lazy}(\sigma_{v_\mathrm{s},v}, \mathcal{E}^{i+1}_\mathrm{eval,GLS}) = 1$, i.e. the subpath from the start to any vertex is valid according to the lazy estimate.

By definition, the heuristic $h_\mathcal{G}(v, v_\mathrm{t}) = w(\sigma_{v,v_\mathrm{t}})$ is the weight of the shortest path on the unevaluated graph σ_{v,v_t}. The heuristic progress threshold h_min is by definition the minimum heuristic value of the child vertex of any evaluated edge, i.e. $h_\mathrm{min} = \min_{(u',v') \in \mathcal{E}^{i+1}_\mathrm{eval,GLS}} h_\mathcal{G}(v', v_\mathrm{t})$. Since $h_\mathcal{G}(v, v_\mathrm{t})$ is consistent, $h_\mathcal{G}(v, v_\mathrm{t}) < h_\mathrm{min} < \min_{(u',v') \in \mathcal{E}^{i+1}_\mathrm{eval,GLS}} h_\mathcal{G}(v', v_\mathrm{t})$ implies that none of the edges $(u', v') \in \sigma_{v,v_\mathrm{t}}$ belong to $\mathcal{E}^{i+1}_\mathrm{eval,GLS}$ have been evaluated. This means that the subpath to goal is valid according to the lazy estimate, i.e. $\phi_\mathrm{lazy}(\sigma_{v,v_\mathrm{t}}, \mathcal{E}^{i+1}_\mathrm{eval,GLS}) = 1$.

[2]See (Mandalika, Salzman, and Srinivasa 2018) for the formal computational model

Hence (4) can be re-written as

$$
\begin{aligned}
v^{i+1} \leftarrow \arg\min_{v} \quad & w(\sigma_{v_{\rm s},v}) + w(\sigma_{v,v_{\rm t}}) \\
\text{s.t.} \quad & \phi_{\rm lazy}(\sigma_{v_{\rm s},v}, \mathcal{E}^{i+1}_{\rm eval,GLS}) = 1 \\
& \phi_{\rm lazy}(\sigma_{v,v_{\rm t}}, \mathcal{E}^{i+1}_{\rm eval,GLS}) = 1
\end{aligned} \tag{5}
$$

Since $\mathcal{E}^{i+1}_{\rm eval,GLS} = \mathcal{E}^{i+1}_{\rm eval,LSP}$, (3) and (5) are the same optimization. Hence $\sigma_{v_{\rm s},v^{i+1}} \subseteq \sigma^{i+1}$ and (B) holds. As a result, the induction holds.

This process continues till both algorithms discover the shortest feasible path σ^* at the end of iteration N. Both evaluate the same number of edges $\mathcal{E}^{N+1}_{\rm eval,LSP} = \mathcal{E}^{N+1}_{\rm eval,GLS}$. But GLS saves on more vertices being rewired than LazySP, i.e. $|\mathcal{V}^N_{\rm rwr,GLS}| \leq |\mathcal{V}^N_{\rm rwr,LSP}| - \sum_{i=1}^{N} |\sigma_{v^i,v_{\rm t}}|$. □

Corollary 4.1. *There is a graph $\mathcal{G}$ for which the number of vertex rewires $\mathcal{V}_{\rm rwr}$ for LazySP over GLS is linear over logarithmic.*

Proof

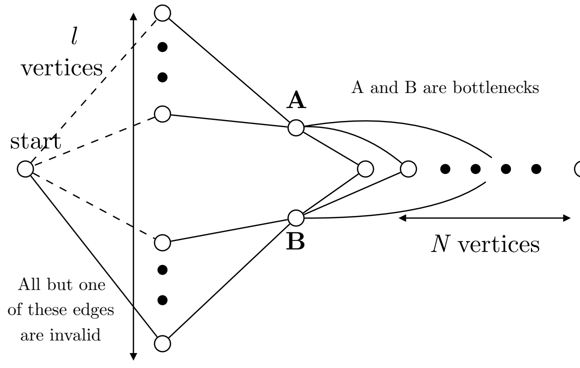

Figure 3: Counter examples

We are going to construct a counter example that shows a particularly bad case of vertex rewiring undertaken by LazySP.

Scenario. Consider the graph in Fig. 3. It has a set of l vertices connected to the start. The upper half of the l vertices are connected to vertex A. The lower half is connected to B. Each of A and B is connected to a chain of N vertices going to the goal. $|\mathcal{V}| = N + l$, $|\mathcal{E}| = 3N + 2l - 1$.

The graph is such that one of l edges connected to the start is valid, the rest is invalid. The remaining edges are all valid.

The weights of the graph are such that the shortest path alternates between the top and bottom halves of the graph. Assume that all $l - 1$ shortest paths are invalid and the last one is valid. Finally assume that A and B alternate being the optimal parent to the N vertices.

LazySP computation .

For LazySP, the only computation is vertex rewiring. The graph is such that successive shortest paths alternate between the upper and lower halves. The shortest paths in the upper half pass through A and lower half pass through B. Hence every edge that is invalidated, causes all N vertices to rewire to either vertex A or B. This is the optimistic thrashing scenario explained in LazyPRM*. Since l edges have to be invalidated, the number of rewires is $O(Nl)$

GLS computation There are two computation steps to account for - heuristic computation and vertex rewiring.

The heuristic computation is a Djikstra operation.

$$
\begin{aligned}
& O(|\mathcal{V}| \log |\mathcal{V}| + |\mathcal{E}|) \\
& O((N + l) \log(N + l) + 3N + 2l - 1) \\
& O((N + l) \log(N + l))
\end{aligned} \tag{6}
$$

Since the search never proceeds beyond the first set of edges, the amount of vertex rewiring is 0.

Hence the complexity of LazySP is $O(Nl)$ while GLS is $O((N + l) \log(N + l))$. The ratio is linear over logarithmic growth.

□

5 Leveraging Edge Priors in GLS

The GLS framework is powerful because one can optimize Event and Selector to minimize computational costs while still retaining guarantees. Here, we show its expressive power in a scenario where we have additional side information, such as priors on the validity of edges. Such information can be collected from datasets of prior experience or generated from approximations of the world representation.

5.1 Modified Problem Formulation

We assume that the validity of each edge is an independent Bernoulli random variable. We are given a vector of probabilities $\mathbf{p} \in [0, 1]^{|\mathcal{E}|}$, such that $P(\phi(e) = 1) = \mathbf{p}(e)$, i.e., for each edge e, we have access to $\mathbf{p}(e)$, which defines the probability of the edge being valid in the current world ϕ.

We allow the shortest path algorithm $\textsc{Alg}(\mathcal{G}, v_{\rm s}, v_{\rm t}, \phi, \mathbf{p})$ to leverage knowledge of edge probabilities $\mathbf{p}$ to minimize the expected computation cost as follows:

$$
\begin{aligned}
\min \quad & \mathbb{E}_{\phi \sim \mathbf{p}} \left[C(\mathcal{E}_{\rm eval}, \mathcal{V}_{\rm rwr}) \right] \\
\text{s.t.} \quad & \mathcal{E}_{\rm eval},\, \mathcal{V}_{\rm rwr} = \textsc{Alg}(\mathcal{G}, v_{\rm s}, v_{\rm t}, \phi, \mathbf{p})
\end{aligned} \tag{7}
$$

5.2 Event and Selector Design

Event. The Event restricts lazy search from proceeding beyond a point when the search is likely to be ineffective, i.e. to a point that potentially increases the amount of rewires $\mathcal{V}_{\rm rwr}$. One such case is when the current shortest subpath is likely to be in collision, i.e., the probability of being valid drops below a threshold δ. We describe this event, SubpathExistence (SE), in Algorithm 2. We show that we can bound the performance of this event.

Theorem 5.1. *For any Selector, the expected planning time of SubpathExistence (δ) can be upper bounded as:*

$$
K \left(c_e \frac{1}{(1 - \delta)} + c_r \frac{b \log(\delta)}{\log(p_{max})} \right) \tag{8}
$$

where K is the number of shortest-paths that are infeasible, b is the maximum branching factor, and p_{max} is the maximum value of an edge prior.

Proof We first describe the GLS algorithm with SubpathExistence (δ). The algorithm searches till the probability of the current shortest subpath drops below δ. It toggles edge evaluation which will either eliminate the subpath or check an edge such that the probability rises above δ. The search continues forward. This repeats till the shortest path has been found. For this proof, we assume we have an oracular selector that can invalidate a subpath if it is truly invalid.

We begin by upper bounding the number of edge evaluations. Let σ^* be the shortest feasible path. Let there be K shorter paths than σ^* that are infeasible and that the algorithm has to eliminate. Since we are showing an upper bound, we can relax the condition that the paths have overlapping edges since they will only reduce edge evaluations (eliminating one implies the other is eliminated).

Consider one of K paths that we have to eliminate. If we pick an edge from the subpath, with probability $1-\delta$ we will find a witness that the path is invalid. A selector either invalidates a subpath with probability $1-\delta$ or results in a wasted edge evaluation. This process is repeated till a path is eventually eliminated. The expected number of edge evaluated to eliminate the path is:

$$\begin{aligned}\mathbb{E}_{\mathbf{p}}\left[\mathcal{E}_{\text{eval}}\right] &\leq (1-\delta)+2\delta(1-\delta)+3\delta^2(1-\delta)+\dots \\ &\leq (1-\delta)\left(1+2\delta+3\delta^2+\dots\right) \\ &\leq (1-\delta)\frac{1}{(1-\delta)^2} \\ &\leq \frac{1}{(1-\delta)}\end{aligned} \tag{9}$$

Hence the total expected cost of edge evaluation is bounded by $c_e K \frac{1}{(1-\delta)}$. Note as $\delta \rightarrow 1$, this term goes to ∞. This is backed by the intuition that triggering the event often results in increased edge evaluation.

We will now upper bound the number of vertex rewiring. We assume that the search is using as heuristic the distance on the graph $h_{\mathcal{G}}(v, v_{\text{t}})$. Hence when one of the K subpaths are eliminated, only the vertices of that subpath is rewired. Since we are deriving an upper bound, we will ignore overlap between subpaths (which can only help).

Consider one of K paths that we have to eliminate. Let p_{max} be the maximum probability of an edge being valid. Then the maximum length of any subpath $L(\delta)$ is

$$\begin{aligned}p_{\text{max}}^{L(\delta)} &\geq \delta \\ L(\delta) &\leq \frac{\log(\delta)}{\log(p_{\text{max}})}\end{aligned} \tag{10}$$

When the subpath is eliminated, the rewiring is restricted only to vertices belonging to the subpath. Hence the maximum vertex rewire that can occur is $\mathcal{V}_{\text{rwr}} = bL(\delta)$ where b is the maximum branching factor. A selector either invalidates a subpath with probability $1-\delta$ and results in rewiring or the process continues without any penalty. The expected number of vertices rewired before the path is eliminated can be upper bounded:

$$\begin{aligned}\mathbb{E}_{\mathbf{p}}\left[\mathcal{V}_{\text{rwr}}\right] &\leq (1-\delta)bL(\delta)+\delta(1-\delta)bL(\delta)+\dots \\ &\leq (1-\delta)bL(\delta)\left(1+\delta+\delta^2+\dots\right) \\ &\leq (1-\delta)bL(\delta)\frac{1}{(1-\delta)} \\ &\leq bL(\delta) \\ &\leq \frac{b\log(\delta)}{\log(p_{\text{max}})}\end{aligned} \tag{11}$$

Hence the total expected cost of vertex rewiring is upper bounded by $c_r K b \frac{\log(\delta)}{\log(p_{\text{max}})}$. Note as $\delta \rightarrow 0$, this term goes to ∞. This is backed by the intuition that triggering the event less often results in increased vertex rewiring. ☐

Low values of δ result in lower edge evaluations but more edge rewiring, and vice-versa.

Corollary 5.1. *There exists a critical threshold $\delta \in (0,1)$ that upper bounds the expected computational cost.*

Proof The total expected planning time can be bounded as:

$$\begin{aligned}\mathbb{E}_{\mathbf{p}}\left[C(\mathcal{E}_{\text{eval}}, \mathcal{V}_{\text{rwr}})\right] &= c_e\left|\mathcal{E}_{\text{eval}}\right| + c_r\left|\mathcal{V}_{\text{rwr}}\right| \\ &= c_e K\frac{1}{(1-\delta)} + c_r K \frac{b\log(\delta)}{\log(p_{\text{max}})} \\ &= K\left(c_e\frac{1}{(1-\delta)} + c_r\frac{b\log(\delta)}{\log(p_{\text{max}})}\right)\end{aligned} \tag{12}$$

We will now show that there exists a critical point δ that minimizes this. Solving for that critical point, we have:

$$\begin{aligned}\frac{\partial}{\partial\delta}\left(K\left(c_e\frac{1}{(1-\delta)} + c_r\frac{b\log(\delta)}{\log(p_{\text{max}})}\right)\right) &= 0 \\ \frac{c_e}{(1-\delta)^2} + \frac{c_r}{\log(p_{\text{max}})}\frac{b}{\delta} &= 0 \\ (1-\delta)^2 - \frac{c_e}{bc_r}\log\left(\frac{1}{p_{\text{max}}}\right)\delta &= 0 \\ \delta^2 - \left(\frac{c_e}{bc_r}\log(\frac{1}{p_{\text{max}}})+2\right)\delta + 1 &= 0\end{aligned} \tag{13}$$

Let $\eta = \left(\frac{c_e}{bc_r}\log(\frac{1}{p_{\text{max}}})+2\right)$. The critical point is:

$$\delta = \frac{\eta - \sqrt{\eta^2-4}}{2} \tag{14}$$

When $c_e \approx c_r$, η is close to 2 and $\delta \rightarrow 1$. When $c_e \gg c_r$, we have $\eta \gg 2$. Hence, the critical point is:

$$\begin{aligned}
\delta &= \frac{\eta - \sqrt{\eta^2 - 4}}{2} \\
&= \frac{\eta\left(1 - \sqrt{1 - \frac{4}{\eta^2}}\right)}{2} \\
&= \frac{\eta\left(1 - \left(1 - \frac{4}{2\eta^2}\right)\right)}{2} \\
&= \frac{1}{\eta} \\
&= \frac{1}{\left(\frac{c_e}{bc_r}\log(\frac{1}{p_{\max}}) + 2\right)}
\end{aligned} \tag{15}$$

The critical point is inversely proportional to the ratio $\frac{c_e}{c_r}$. □

Selector. The SELECTOR invalidates as many subpaths as quickly as possible, which restricts the size of $\mathcal{E}_{\text{eval}}$. One strategy for doing so is to invalidate the current subpath as quickly as possible. We describe a selector, FAILFAST (FF), in Algorithm 3 that evaluates the edge on the subpath with the highest probability of being in collision. We show that this selector is the optimal strategy to invalidate a subpath.

Theorem 5.2. *Given a path σ, FAILFAST minimizes the expected number of edges from σ that must be evaluated to invalidate σ.*

Proof Given a path σ, and a sequence of edges $S = \{e_1, e_2, \ldots, e_n\}$ belonging to the path, and the corresponding priors of the edges being valid $(p_1, p_2, \ldots, p_n)$, let the expected number of edge evaluations to invalidate the σ be $\mathcal{E}_{\text{eval}}(S)$ which is given by

$$\begin{aligned}
\mathbb{E}_{\mathbf{p}}\left[\mathcal{E}_{\text{eval}}(S)\right] &= (1 - p_1) + 2p_1(1 - p_2) + \ldots \\
&= \sum_{l=1}^{n}\left(\prod_{m=1}^{l-1} p_m\right)(1 - p_l)\, l
\end{aligned} \tag{16}$$

Without loss of generality, let $p_i > p_{i+1}$ for a given i. Consider the alternate sequence of evaluations $S' = \{e_1, e_2, \ldots, e_{i+1}, e_i \ldots, e_n\}$ where the positions of the edges $e_i,\ e_{i+1}$ are swapped. Consider the difference:

$$\begin{aligned}
&\mathbb{E}_{\mathbf{p}}\left[\mathcal{E}_{\text{eval}}(S)\right] - \mathbb{E}_{\mathbf{p}}\left[\mathcal{E}_{\text{eval}}(S')\right] \\
&= \ldots + \prod_{m=1}^{i-1} p_m\left[(1 - p_i)i + p_i(1 - p_{i+1})(i+1)\right] + \ldots \\
&- \ldots + \prod_{m=1}^{i-1} p_m\left[(1 - p_{i+1})i + p_{i+1}(1 - p_i)(i+1)\right] + \ldots \\
&= \prod_{m=1}^{i-1} p_m\left[-i(p_i - p_{i+1}) + (i+1)(p_i - p_{i+1})\right] \\
&= \prod_{m=1}^{i-1} p_m(p_i - p_{i+1}) \\
&> 0
\end{aligned} \tag{17}$$

Since each such swap results in monotonic decrease in the objective, there exists an unique fixed point, i.e., the optimal sequence S^* has $p_1 \leq p_2 \leq \ldots \leq p_n$. □

5.3 Hypotheses

Based on our theoretical analysis and insight, we state three hypotheses that we intend to test:

H 1. *For any SELECTOR, the event SUBPATHEXISTENCE requires less planning time compared to SHORTESTPATH and CONSTANTDEPTH.*

This follows from Theorem 5.1, which upper bounds the planning time for SUBPATHEXISTENCE. SHORTESTPATH corresponds to $\delta = 0$ and can increase planning time. CONSTANTDEPTH has a fixed lookahead and does not adapt as priors change.

H 2. *For any EVENT, FAILFAST evaluates fewer edges than FORWARD and ALTERNATE.*

This follows from Theorem 5.2, which shows that FAILFAST is optimal in expectation for eliminating a path. From **H 1** and **H 2**, we hypothesize that the combination of SUBPATHEXISTENCE and FAILFAST will have the lowest planning time.

H 3. *The performance gain of SUBPATHEXISTENCE over SHORTESTPATH increases with both graph size and problem difficulty.*

SHORTESTPATH assumes that $\mathcal{V}_{\text{rwr}}$ is negligible. As graph size increases, the size of vertices $\mathcal{V}_{\text{rwr}}$ that SHORTESTPATH rewires also increases. Similarly, as problem difficulty increases, so does the number of shortest paths that SHORTESTPATH must invalidate, which also increases $\mathcal{V}_{\text{rwr}}$. SUBPATHEXISTENCE, on the other hand, makes no such assumption.

6 Experiments

Algorithm Details. We implemented 3 EVENTS and 3 SELECTORS described in Algorithms 2 and 3 to get a total of 9 algorithms. To analyze the trade-offs, we test on a diverse set of $\mathbb{R}^2$ datasets. We then finalize on 3 algorithms: `LazySP` (SHORTESTPATH, FAILFAST), `LRA*` (CONSTANTDEPTH, FAILFAST) and `GLS` (SUBPATHEXISTENCE, FAILFAST). We evaluate these on a Piano Movers' problem in $SE(2)$ and manipulation problems in $\mathbb{R}^7$ using HERB (Srinivasa et al. 2009), a mobile robot with 7DoF arms. [3]

Analysis on $\mathbb{R}^2$ datasets. We use 5 datasets of $\mathbb{R}^2$ problems from (Choudhury et al. 2017). Each dataset corresponds to different parametric distribution of obstacles from which we sample 1000 worlds. A graph of 2000 vertices is sampled using a low dispersion sampler (Halton 1964) with an optimal connection radius (Janson et al. 2015). Priors are computed by collision checking the graph on the training data and averaging edge outcomes. The prior and some samples from $\mathbb{R}^2$ datasets are shown in Fig. 4(a). We pick one representative dataset, TwoWall, to show detailed plots.

[3]Code is publicly available as an OMPL Planner at: https://github.com/personalrobotics/gls

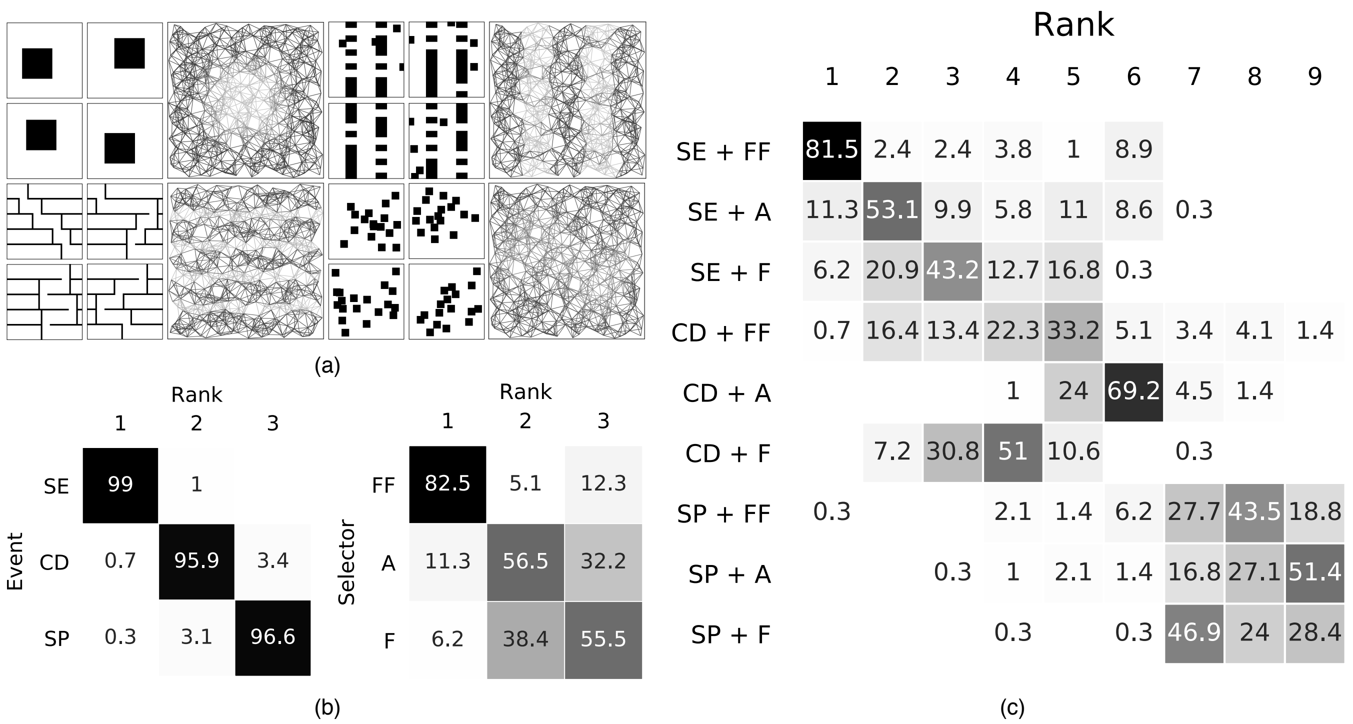

Figure 4: (a) Samples and the prior for $\mathbb{R}^2$ datasets. (b) Events, Selectors and (c) Algorithms, ranked by planning times on $\mathbb{R}^2$ environments. Each cell indicates percentage of problems on which corresponding rank has been obtained.

	`SP + F`	`SP + A`	`SP + FF`	`CD + F`	`CD + A`	`CD + FF`	`SE + F`	`SE + A`	`SE + FF`
Square									
Total Planning Time	0.331	0.454	0.372	0.221	0.259	0.222	0.171	0.161	**0.116**
# Edge Evaluations	190	308	137	273	344.5	315.5	200.5	206	153
# Vertex Rewires	7058.5	8502.5	9859	1076.5	641.5	69.5	1104.5	603.5	318.5
Two Wall									
Total Planning Time	0.419	0.394	0.377	0.262	0.301	0.290	0.220	0.169	**0.161**
# Edge Evaluations	224	242.5	144	310	393	407	287	224.5	202.5
# Vertex Rewires	9360	7997	9870	1594.6	914.5	165.5	697	435	711.5
Mazes									
Total Planning Time	1.334	1.292	1.272	0.574	0.776	0.560	0.615	0.578	**0.337**
# Edge Evaluations	531	471	307.5	630	895	785.5	588	544.5	352.5
# Vertex Rewires	34359	34379.5	37750	4769	5357.5	326	7266.5	7039.5	3213
Forest									
Total Planning Time	0.277	0.267	0.269	0.574	0.776	0.559	**0.219**	0.234	0.229
# Edge Evaluations	174.5	184.5	165.5	306.5	342.5	450.5	190	220	174.5
# Vertex Rewires	5524.5	4936	5467.5	579	346	95.5	3075	2855	3827.5

Table 2: Planning Time(millisec.) and number of operations by algorithms under `GLS`. (Median reported on 100 tests.)

We choose evaluation metrics (a) number of edge evaluations (b) number of vertex rewires and c) total planning time: weighted combination of (a), (b) (see Eq. 2). Since $\mathbb{R}^2$ problems are *not expensive to evaluate*, we choose weights based on empirical data from manipulation planning problems in $\mathbb{R}^7$ (avg. eval time: 3.35×10^{-4}s, avg. rewire time 1.1×10^{-5}s, ratio 29.04).

Finally, for parameter selection, we choose δ in SubpathExistence from the pareto curve of vertices rewired vs edges evaluated computed on the training data. The slope of the line is the ratio of their relative cost – the point of intersection corresponds to the $\delta : 0.01$ that minimizes planning time. For ConstantDepth, we use the recommended value from (Mandalika, Salzman, and Srinivasa 2018).

Table 2 shows the planning times of various algorithms under `GLS`. The planning times are the median quantities

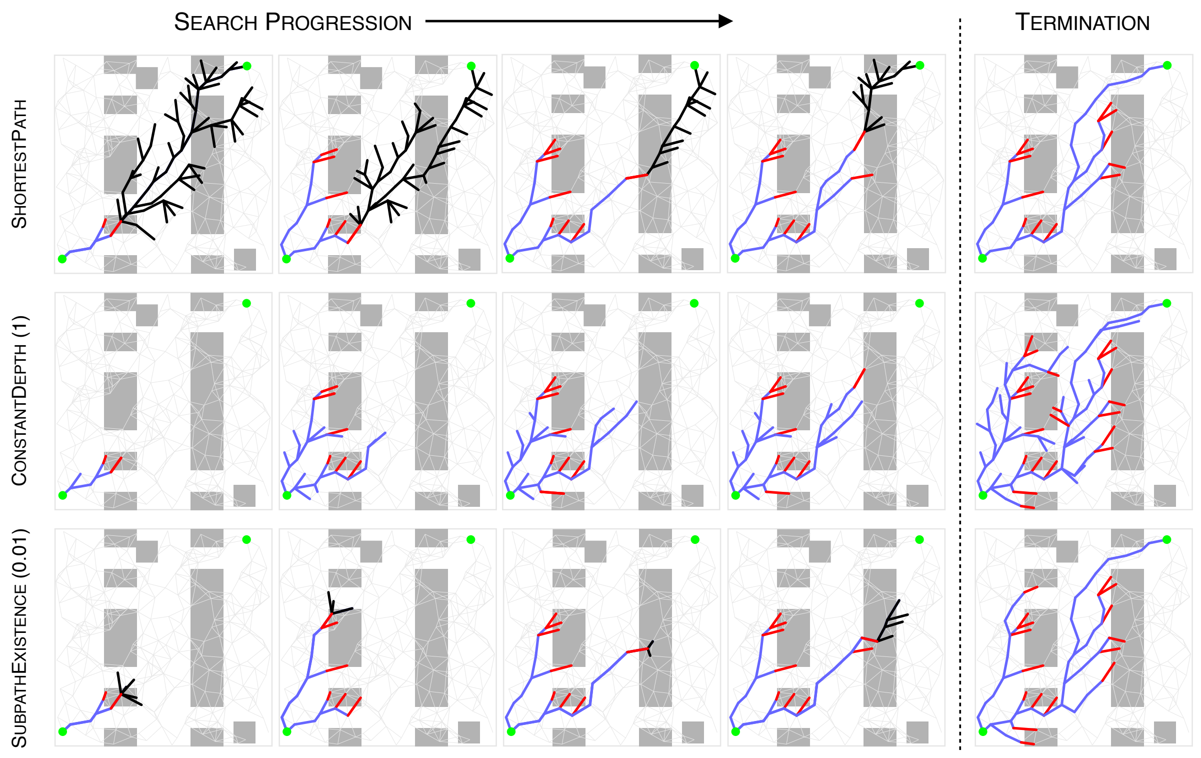

Figure 5: Snapshots of search and evaluation by GLS with FORWARD selector and different events. Edges evaluated to be valid (blue), invalid (red) and subtree of vertices to be rewired (black) are shown. From top to bottom at termination: the number of edge evaluations are (49, 97, 62) and the number of vertex rewires are (361, 21, 69).

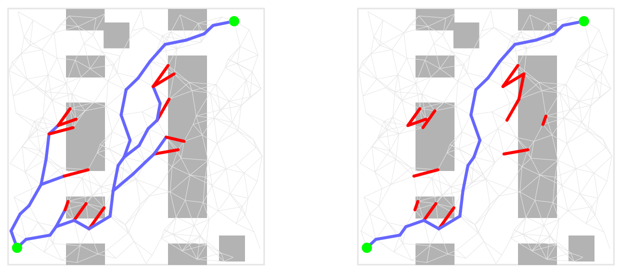

Figure 6: All valid (blue) and invalid (red) edges evaluated at termination of GLS with FORWARD (49 edges) and FAILFAST (32 edges) with SUBPATHEXISTENCE.

obtained from experiments over 100 different worlds sampled within the environment type. Fig. 4(c) shows the ranking of the planning times of the algorithms across the 400 worlds considered across the four datasets (lower plannning time in a problem translates to a better rank). We note that GLS with SUBPATHEXISTENCE and FAILFAST consistently outperforms remaining algorithms on a majority of environments.

We found strong evidence to support **H 1** - SUBPATHEXISTENCE exhibits lowest planning times in 99% of the problems (Fig. 4(b, left)) Corresponding median planning times supporting the hypothesis are reported in Table 2. Fig. 5 shows a comparison of the events SHORTESTPATH, CONSTANTDEPTH and SUBPATHEXISTENCE (for the FORWARD selector) on a problem from the TwoWall dataset. We can see that SHORTESTPATH checks small number of edges but rewires significant portion of the search tree. The trend is reversed in CONSTANTDEPTH when using a depth of 1. SUBPATHEXISTENCE is able to balance both by exploiting priors - it triggers events when the search reaches the walls thus reducing rewires.

We also found strong evidence to support **H 2** - FAILFAST exhibits the lowest planning times in 83% (Fig. 4(b, right)) of the problems across the four datasets. In Table 2, we note that for a given event, FAILFAST has the lowest planning time in majority of the datasets. Fig. 6 shows a comparison of FORWARD and FAILFAST (for SHORTESTPATH event) - FAILFAST quickly eliminates paths by checking the weakest link (supporting Theorem 5.2).

We found strong evidence to support **H 3**. Fig. 7b shows that as graphs get larger, planning times of SHORTESTPATH grows at a faster rate than SUBPATHEXISTENCE. Fig. 7c shows that as the density of obstacles increase, the planning times of SHORTESTPATH grows linearly while SUBPATHEXISTENCE eventually saturates.

Analysis on $SE(2)$ problems and $\mathbb{R}^7$ problems. We consider the Piano Movers' problem in $SE(2)$ from the Apartment scenario in OMPL (Şucan, Moll, and Kavraki 2012). For the $\mathbb{R}^7$ environment, we consider two manipulation tasks

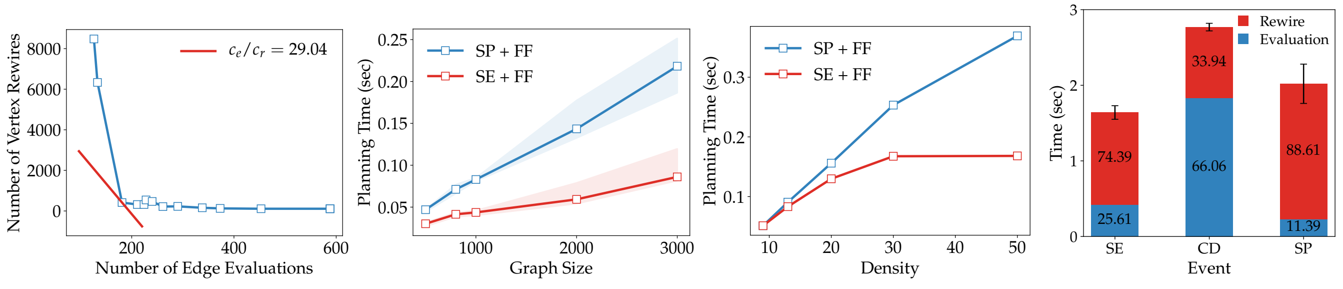

Figure 7: (a) Pareto curve obtained by varying δ in SUBPATHEXISTENCE (b) Planning times as the size of the graph in TwoWall is increased. (c) Planning time as the density of obstacles in Forest is increased (d) Planning time in $\mathbb{R}^7$ problem (with 95% C.I.)

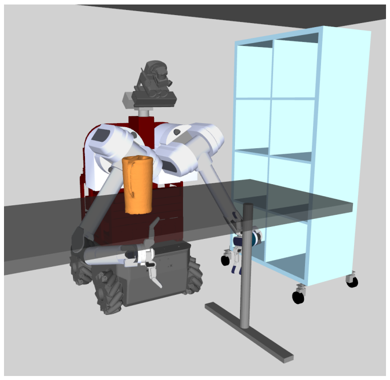

Figure 8: HERB Task 1: Robot reaches into the shelf with its right arm while avoiding the table and the object.

Table 3: Mean Planning Times (in seconds) of `GLS`, `LazySP` and `LRA*` on $SE(2)$, $\mathbb{R}^7$ problems.

	`GLS`	`LRA*`	`LazySP`
Piano Movers'			
Total Planning Time	**1.00**	1.13	1.25
Edge Evaluation Time	0.17	0.49	0.10
Vertex Rewire Time	0.83	0.64	1.15
HERB Task 1			
Total Planning Time	**1.17**	1.81	1.53
Edge Evaluation Time	0.38	1.19	0.29
Vertex Rewire Time	0.79	0.62	1.24
HERB Task 2			
Total Planning Time	**1.64**	2.77	2.02
Edge Evaluation Time	0.42	1.83	0.23
Vertex Rewire Time	1.22	0.94	1.79

with a 7-DoF arm (Srinivasa et al. 2009) in a cluttered kitchen environment (Fig. 8). We used graphs of 8000 vertices and 30,000 vertices for the $SE(2)$ and $\mathbb{R}^7$ problems respectively.

In Table 3, we report the mean planning times across 100 problems each. We see that `GLS`(SUBPATHEXISTENCE, FAILFAST) outperforms the other algorithms in planning time on all three problems. Additionally, Fig. 7d shows a breakdown of the planning time for each of the three events on *HERB Task 2*. `GLS` significantly lowers rewiring time while having a minimal increase in evaluation time.

Figures 9, 10 compare the performance of `LazySP` and `GLS` with FAILFAST selector. They illustrate the savings of `GLS` on the Piano Movers' problem (Fig. 9) and on a simplified manipulation scene (Fig. 10). In both cases, `LazySP` has to rewire a large search tree everytime a path is found to be in collision. `GLS`, on the other hand, halts the search as soon as it enters a region of low probability, eliminates the paths and hence drastically minimizes rewiring time at the cost of few additional edge evaluations over `LazySP`.

7 Discussion

We presented a general framework for lazy search (`GLS`). The staple framework interleaves two phases, search and evaluation. In the search phase, it extends a lazy shortest-path tree forward without evaluating any edges until an EVENT is triggered. It then switches to evaluation phase. It finds the shortest subpath to a leaf node of the tree and invokes a SELECTOR to evaluate an edge on it. Careful choice of EVENT and SELECTOR allows the balance of search effort with edge evaluation to minimize overall planning time.

The framework, quite expressive, lets us capture a range of lazy search algorithms (Table 1). While it draws inspiration from prior work interleaving search and evaluation, such as `LRA*` (Mandalika, Salzman, and Srinivasa 2018), the key difference lies in our definition of the EVENT, which makes the algorithm *adaptive*. This lets us derive new algorithms that are edge optimal while saving on search effort (Theorem 4.3).

In future work, we plan to examine more sophisticated SELECTOR policies (Choudhury, Srinivasa, and Scherer 2018) that exploit correlations amongst edges to minimize evaluation cost. We also plan to extend `GLS` to an anytime paradigm; this would let us use heuristics that exploit edge priors to guide the search through regions of high probability (Nielsen and Kavraki 2000), for significant speed-ups. Finally, we plan to explore problems where multiple lazy estimates of weight functions are available, e.g., in kinodynamic planning, where different relaxations of the boundary value problem can be obtained. We believe `GLS` can interleave search efficiently over multiple resolutions of approximation.

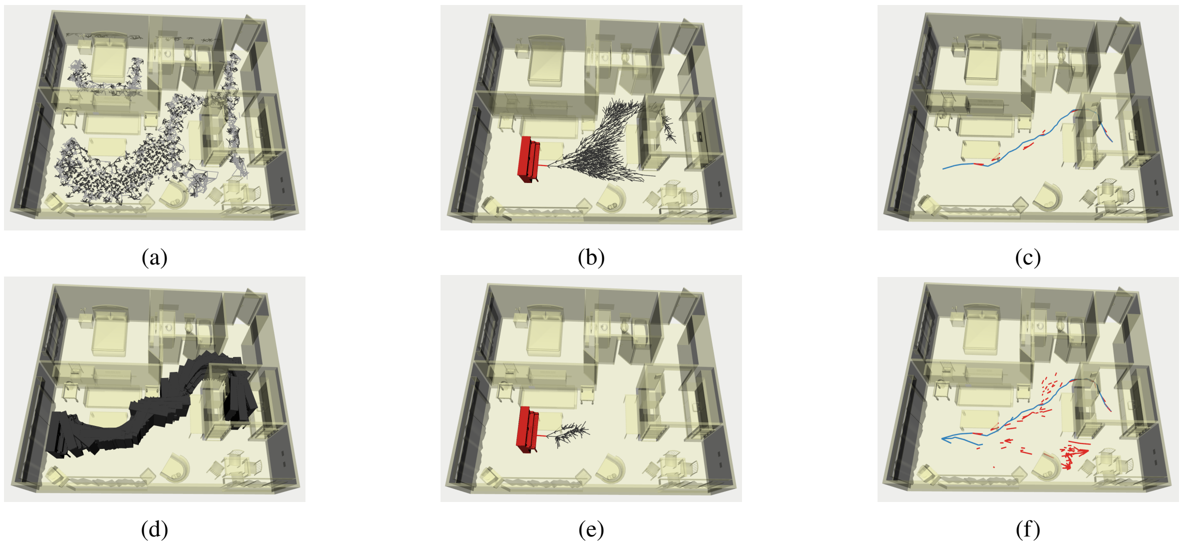

Figure 9: (a) Edge priors: darker edges have higher prior. (d) Solution path on the graph. Second and Third columns visualize the search and evaluation by LazySP(top) and GLS(SubpathExistence) (bottom). (b) and (e): subtree of vertices rewired in the first iteration (12,495 and 1235 resp. at termination). (c) and (f): edges evaluated at termination (63 and 171 resp.).

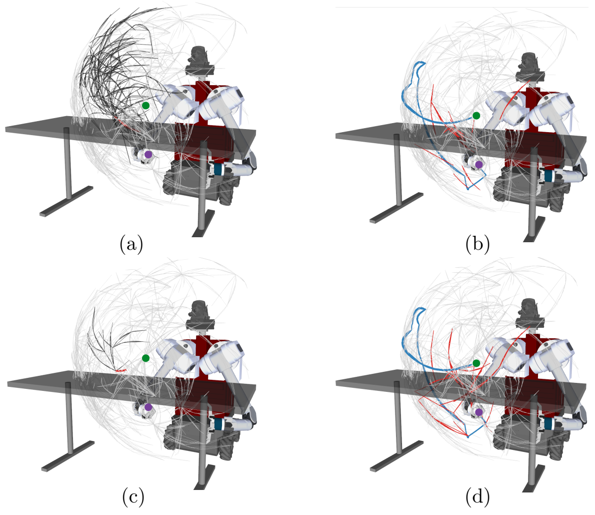

Figure 10: Search and evaluation by LazySP (top) and GLS(SubpathExistence) (bottom). (a), (c): subtree of vertices rewired in the first iteration (21,178; 11,342 resp. at termination). (b), (d): edges evaluated (136; 243 resp.).

References

Bialkowski, J.; Otte, M. W.; Karaman, S.; and Frazzoli, E. 2016. Efficient collision checking in sampling-based motion planning via safety certificates. *I. J. Robotics Res.* 35(7):767–796.

Bialkowski, J.; Otte, M.; and Frazzoli, E. 2013. Free-configuration biased sampling for motion planning. In *IROS*, 1272–1279. IEEE.

Bohlin, R., and Kavraki, L. E. 2000. Path planning using lazy PRM. In *ICRA*, volume 1, 521–528. IEEE.

Burns, B., and Brock, O. 2005. Sampling-based motion planning using predictive models. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 3120–3125. IEEE.

Choudhury, S.; Javdani, S.; Srinivasa, S.; and Scherer, S. 2017. Near-optimal edge evaluation in explicit generalized binomial graphs. In *NIPS*, 4634–4644.

Choudhury, S.; Dellin, C. M.; and Srinivasa, S. S. 2016. Pareto-optimal search over configuration space beliefs for anytime motion planning. In *IROS*, 3742–3749.

Choudhury, S.; Srinivasa, S.; and Scherer, S. 2018. Bayesian Active Edge Evaluation on Expensive Graphs. In *IJCAI*, 4890–4897.

Cohen, B. J.; Phillips, M.; and Likhachev, M. 2014. Planning Single-arm Manipulations with n-Arm Robots. In *RSS*.

Dellin, C. M., and Srinivasa, S. S. 2016. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *ICAPS*, 459–467.

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

Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2015. Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *ICRA*, 3067–3074.

Haghtalab, N.; Mackenzie, S.; Procaccia, A. D.; Salzman, O.; and Srinivasa, S. S. 2018. The Provable Virtue of Laziness in Motion Planning. In *ICAPS*, 106–113.

Halton, J. H. 1964. Algorithm 247: Radical-inverse quasi-random point sequence. *Commun. ACM* 7(12):701–702.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Hauser, K. 2015. Lazy collision checking in asymptotically-optimal motion planning. In *ICRA*, 2951–2957.

Huh, J., and Lee, D. D. 2016. Learning high-dimensional mixture models for fast collision detection in rapidly-exploring random trees. In *ICRA*.

Janson, L.; Schmerling, E.; Clark, A. A.; and Pavone, M. 2015. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *I. J. Robotics Res.* 34(7):883–921.

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

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics and Automation* 12(4):566–580.

Kim, D.; Kwon, Y.; and Yoon, S.-e. 2018. Adaptive lazy collision checking for optimal sampling-based motion planning. In *UR*, 320–327. IEEE.

Koenig, S., and Sun, X. 2009. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems* 18(3):313–341.

Koenig, S.; Likhachev, M.; and Furcy, D. 2004. Lifelong planning A*. *Artif. Intell.* 155(1-2):93–146.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97 – 109.

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2):189 – 211.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2004. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in neural information processing systems*, 767–774.

Mandalika, A.; Salzman, O.; and Srinivasa, S. 2018. Lazy Receding Horizon A* for Efficient Path Planning in Graphs with Expensive-to-Evaluate Edges. In *ICAPS*, 476–484.

Murray, S.; Floyd-Jones, W.; Qi, Y.; Sorin, D. J.; and Konidaris, G. 2016. Robot motion planning on a chip. In *RSS*.

Narayanan, V., and Likhachev, M. 2017. Heuristic search on graphs with existence priors for expensive-to-evaluate edges. In *ICAPS*.

Nielsen, C. L., and Kavraki, L. E. 2000. A 2 level fuzzy prm for manipulation planning. In *IROS*.

Phillips, M.; Cohen, B.; Chitta, S.; and Likhachev, M. 2012. E-graphs: Bootstrapping planning with experience graphs. In *Proceedings of Robotics: Science and Systems*.

Salzman, O., and Halperin, D. 2015. Asymptotically-optimal Motion Planning using lower bounds on cost. In *ICRA*, 4167–4172.

Salzman, O., and Halperin, D. 2016. Asymptotically Near-Optimal RRT for Fast, High-Quality Motion Planning. *IEEE Trans. Robotics* 32(3):473–483.

Srinivasa, S. S.; Ferguson, D.; Helfrich, C. J.; Berenson, D.; Collet, A.; Diankov, R.; Gallagher, G.; Hollinger, G.; Kuffner, J.; and Weghe, M. V. 2009. HERB: a home exploring robotic butler. *Autonomous Robots* 28(1):5.

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

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with partial expansion for large branching factor problems. In *AAAI*, 923–929.