Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

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.

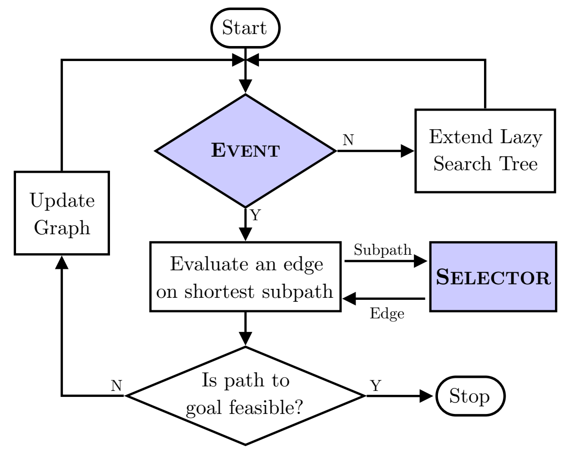

Figure 1: The Generalized Lazy Search (`GLS`) framework with two parameters - Event and Selector (blue)

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 different Event and Selector pairs, we can recover several existing lazy search algorithms such as `LazySP` (Dellin

*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.

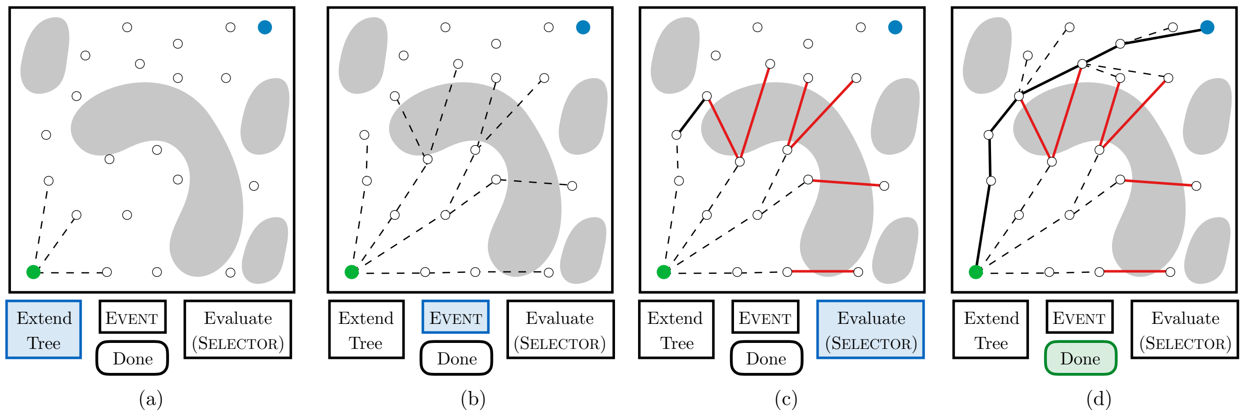

Figure 2: Mechanics of the GLS framework (Algorithm 1) for an ideal Event and Selector combination.

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 $(v_1, v_2, \ldots, v_l)$ 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. $\textsc{Alg}$ 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_s, target v_t, world ϕ
Parameter : $\textsc{Event}$, $\textsc{Selector}$
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 $\quad \mathcal{T}_\mathrm{lazy} \leftarrow \textsc{ExtendTree}\ (\textsc{Event}, \mathcal{T}_\mathrm{lazy})$ ▷ Add $\mathcal{V}_\mathrm{rwr}$
5 $\quad \sigma_\mathrm{sub} \leftarrow \textsc{GetShortestPathToLeaf}\ (\mathcal{T}_\mathrm{lazy})$
6 $\quad \textsc{EvaluateEdge}\ (\textsc{Selector}, \sigma_\mathrm{sub})$ ▷ 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 $\textsc{Alg}$. 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: $\textsc{Event}$ and $\textsc{Selector}$.

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 $\textsc{ExtendTree}$ additionally takes as input a function $\textsc{Event}$. Extending $\mathcal{T}_\mathrm{lazy}$ triggers the $\textsc{Event}$ 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 $\textsc{Event}$ acts as a toggle between lazy seach and edge evaluation.

Definition 4.1 ($\textsc{Event}$)**.** *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 $\textsc{EvaluateEdge}$ also takes as input a function $\textsc{Selector}$. The $\textsc{Selector}$ acts on σ_sub and returns an edge belonging to it that the algorithm evaluates.

Definition 4.2 ($\textsc{Selector}$)**.** *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 $v = v_{\text{t}}$ **then**
    return true;
  end
Function CONSTANTDEPTH(*depth* α)
  $\sigma_{\text{sub}} \leftarrow$ path from v_{s} to v
  $\alpha_v \leftarrow$ number of unevaluated edges in σ_{sub}
  if $\alpha_v = \alpha$ ***or*** $v = v_{\text{t}}$ **then**
    return true;
  end
Function HEURISTICPROGRESS
  $h_{\text{min}} \leftarrow \min_{(u',v') \in \mathcal{E}_{\text{eval}}} h(v', v_{\text{t}})$
  if $h(v, v_{\text{t}}) < h_{min}$ ***or*** $v = v_{\text{t}}$ **then**
    return true;
  end
  return;
Function SUBPATHEXISTENCE(*probability* δ)
  $\sigma_{\text{sub}} \leftarrow$ path from v_{s} to v
  $p \leftarrow \prod_{e \in \sigma} \mathbf{p}(e)$
  if $p \leq \delta$ ***or*** $v = v_{\text{t}}$ **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 $v_{\text{s}}$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., $h_{\text{min}} \leftarrow \min_{(u',v') \in \mathcal{E}_{\text{eval}}} h(v', v_{\text{t}})$. 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)	CONSTANTDEPTH (∞)	Any
LazySP (2016)	CONSTANTDEPTH (∞)	Any
LWA* (2014)	CONSTANTDEPTH (1)	FORWARD
LRA* (2018)	CONSTANTDEPTH (α)	FORWARD

Table 1: Equivalence of GLS and 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.*

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

`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*

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

See (Mandalika et al. 2019) for details and formal proofs.

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

$$\begin{aligned} &\min \;\; \mathbb{E}_{\phi \sim \mathbf{p}} \left[C(\mathcal{E}_\mathrm{eval}, \mathcal{V}_\mathrm{rwr}) \right] \\ &\text{s.t.} \quad \mathcal{E}_\mathrm{eval},\ \mathcal{V}_\mathrm{rwr} = \textsc{Alg}(\mathcal{G}, v_\mathrm{s}, v_\mathrm{t}, \phi, \mathbf{p}) \end{aligned} \tag{3}$$

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 $\mathcal{V}_\mathrm{rwr}$. 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:*

$$K \left(c_e \frac{1}{(1-\delta)} + c_r \frac{b \log(\delta)}{\log(p_{max})} \right) \tag{4}$$

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.

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.*

Selector. The Selector invalidates as many subpaths as quickly as possible, which restricts the size of $\mathcal{E}_\mathrm{eval}$. One strategy for doing so is to invalidate the current subpath as

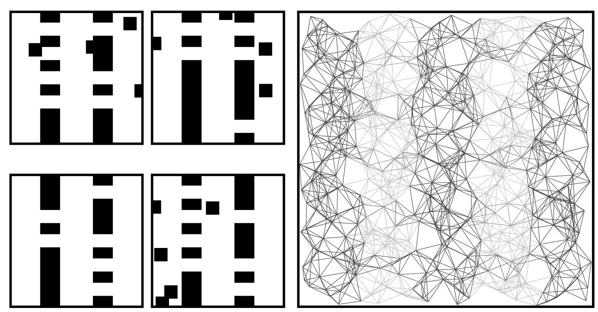

Figure 3: Samples and the prior for the TwoWall dataset.

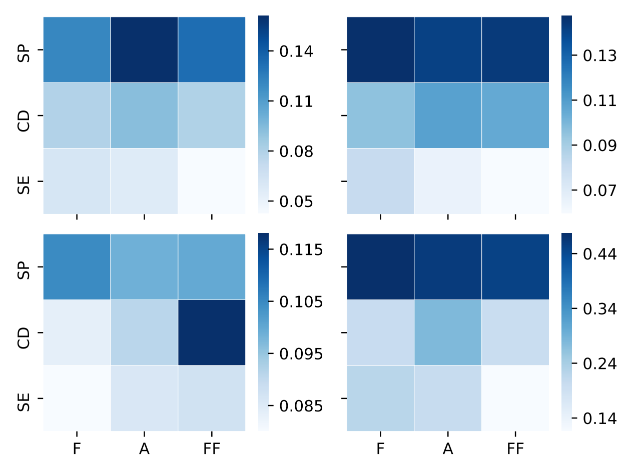

Figure 4: The mean planning times for the 9 algorithms on 4 of the $\mathbb{R}^2$ datasets. Top: Square and TwoWall. Bottom: Forest and Maze. In all except the Forest dataset, SubpathExistence with FailFast performs best.

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

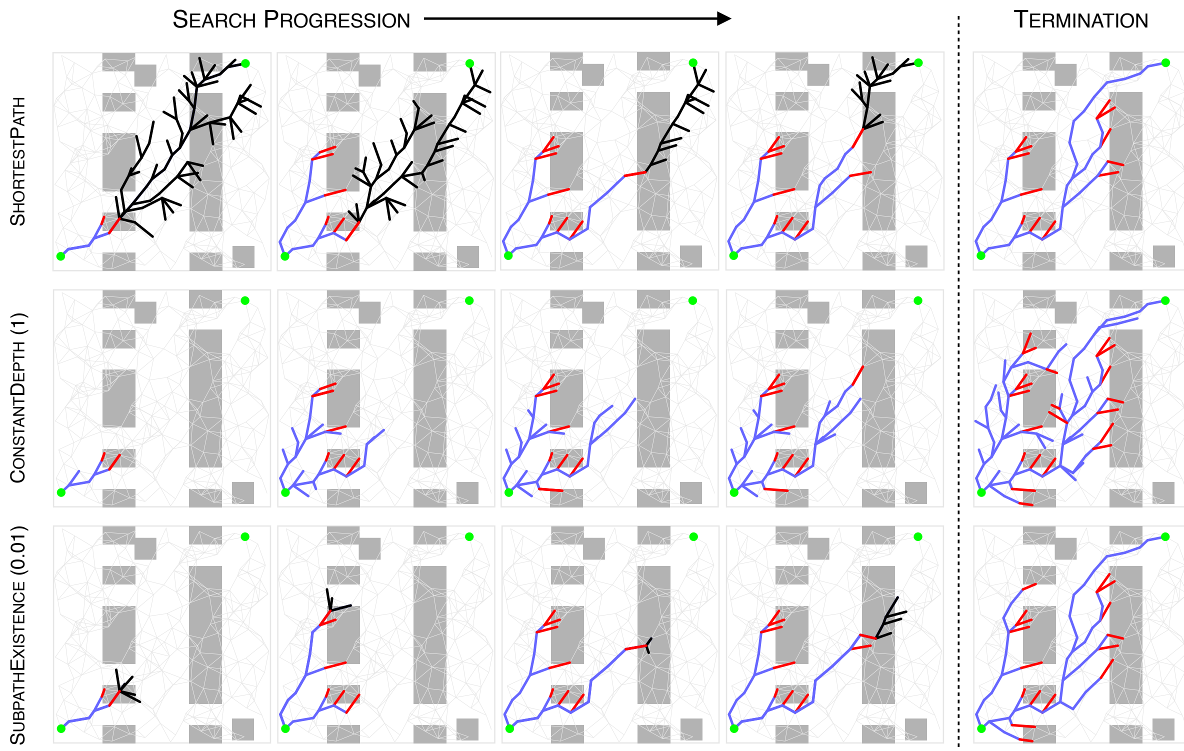

Figure 5: Snapshots of search and evaluation by GLS with FORWARD selector and different events. Edges evaluated to be valid (blue), invalid (red) and subtree of vertices to be rewired (black) are shown. From top to bottom at termination: the number of edge evaluations are (49, 97, 62) and the number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

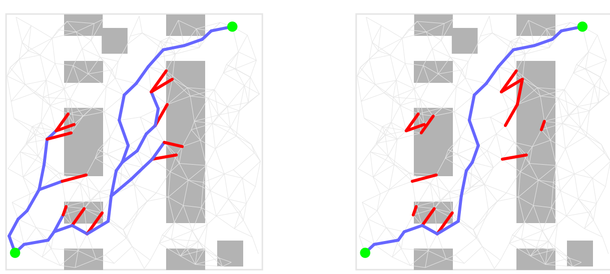

Figure 6: All valid (blue) and invalid (red) edges evaluated at termination of GLS with FORWARD (49 edges) and FAILFAST (32 edges) with SUBPATHEXISTENCE.

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.*

See (Mandalika et al. 2019) for details and formal proofs.

5.3 Hypotheses

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

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

	GLS	LazySP	LRA*
Piano Movers'			
Total Planning Time	**0.595**	0.877	0.742
Edge Evaluation Time	0.129	0.035	0.248
Vertex Rewire Time	0.466	0.842	0.494
HERB Task 1			
Total Planning Time	**1.235**	1.618	2.069
Edge Evaluation Time	0.423	0.388	1.422
Vertex Rewire Time	0.813	1.230	0.647
HERB Task 2			
Total Planning Time	**1.737**	2.100	2.999
Edge Evaluation Time	0.448	0.362	1.967
Vertex Rewire Time	1.292	1.738	1.032

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. CONSTANT-

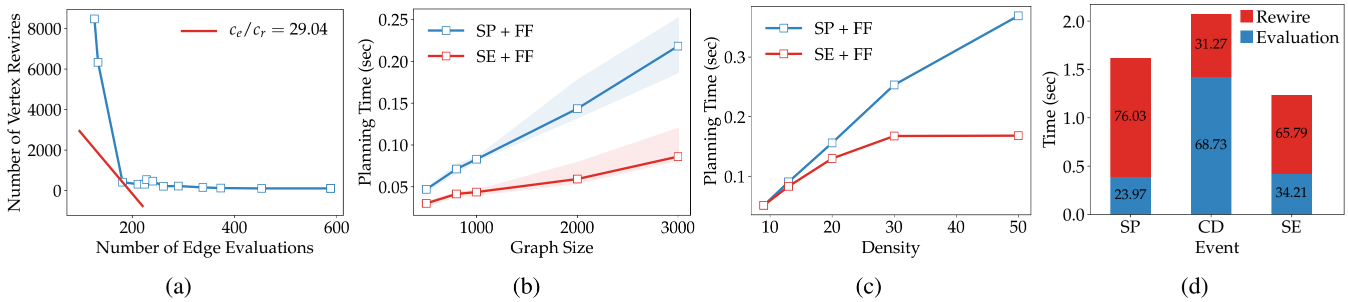

Figure 7: (a) Pareto curve obtained by varying δ in SubpathExistence (b) Planning times as the size of the graph in TwoWall is increased. (c) Planning time as the density of obstacles in Forest is increased (d) Planning time in $\mathbb{R}^7$ problem.

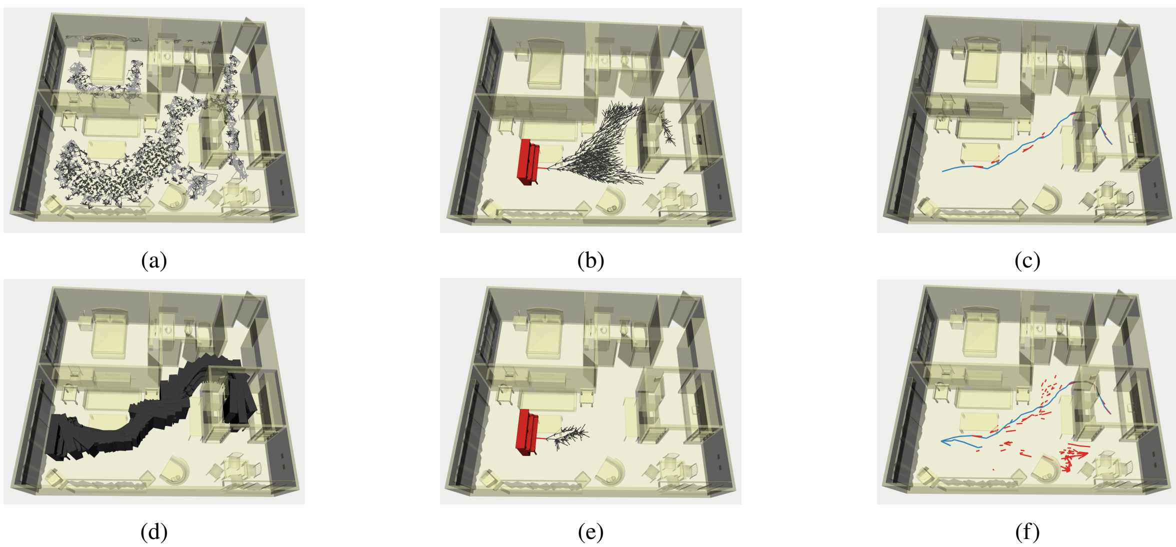

Figure 8: (a) Edge priors: darker edges have higher prior. (d) Solution path on the graph. Second and Third columns visualize the search and evaluation by `LazySP`(top) and `GLS`(SubpathExistence) (bottom). (b) and (e): subtree of vertices rewired in the first iteration (12,495 and 1235 resp. at termination). (c) and (f): edges evaluated at termination (63 and 171 resp.).

Depth 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

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

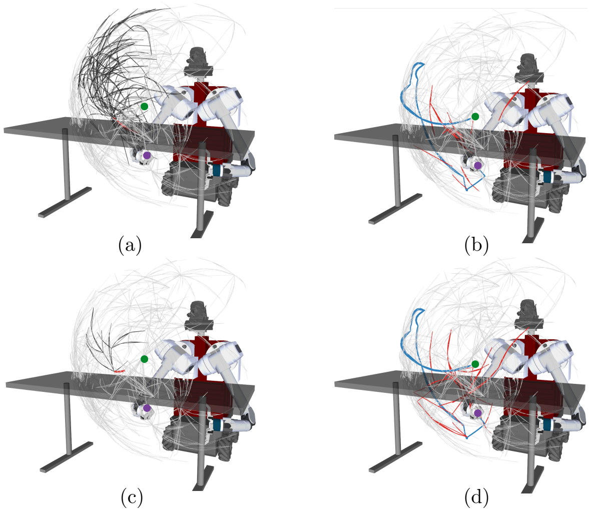

Figure 9: Search and evaluation by `LazySP` (top) and `GLS`(SubpathExistence) (bottom). (a), (c): subtree of vertices rewired in the first iteration (21,178; 11,342 resp. at termination). (b), (d): edges evaluated (136; 243 resp.).

by collision checking the graph on the training data and averaging edge outcomes. We pick one representative dataset, TwoWall, to show detailed plots. The prior and some samples from TwoWall is shown in Fig. 3.

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 interesection corresponds to the $\delta : 0.01$ that minimizes planning time. For ConstantDepth, we use the recommended value from (Mandalika, Salzman, and Srinivasa 2018).

Fig. 4 shows the mean planning time of all 9 algorithms on 4 of the datasets (lighter implies lower planning times). We observe that `GLS`(SubpathExistence, FailFast) outperforms all other possible instantiations of algorithms on majority of datasets. We will use this to analyze each hypothesis in Section. 5.3. We found evidence to support **H 1** - for every column in Fig. 4, SubpathExistence has the lowest planning times. 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 found partial evidence to support **H 2** - FailFast has the lowest planning time for ShortestPath and SubpathExistence. However, the exception was ConstantDepth for the Forest dataset - we attribute this to an artifact of the event dataset combination. 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 with a 7-DoF arm (Srinivasa et al. 2009) in a cluttered kitchen environment. We used graphs of 8000 vertices and 30,000 vertices for the $SE(2)$ and $\mathbb{R}^7$ problems respectively (Mandalika et al. 2019).

In Table 2 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 8, 9 compare the performance of `LazySP` and `GLS` with FailFast selector. They illustrate the savings of `GLS` on the Piano Movers' problem (Fig. 8) and on a simplified manipulation scene (Fig. 9). 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.

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.; Choudhury, S.; Salzman, O.; and Srinivasa, S. 2019. Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles. *arxiv*.

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.