Leveraging Experience in Lazy Search

Mohak Bhardwaj *, Sanjiban Choudhury †, Byron Boots * and Siddhartha Srinivasa †
*Georgia Institute of Technology †University of Washington

***Abstract*—Lazy graph search algorithms are efficient at solving motion planning problems where edge evaluation is the computational bottleneck. These algorithms work by lazily computing the shortest potentially feasible path, evaluating edges along that path, and repeating until a feasible path is found. The order in which edges are selected is critical to minimizing the total number of edge evaluations: a good edge selector chooses edges that are not only likely to be invalid, but also eliminates future paths from consideration. We wish to learn such a selector by leveraging prior experience. We formulate this problem as a Markov Decision Process (MDP) on the state of the search problem. While solving this large MDP is generally intractable, we show that we can compute oracular selectors that can solve the MDP during training. With access to such oracles, we use imitation learning to find effective policies. If new search problems are sufficiently similar to problems solved during training, the learned policy will choose a good edge evaluation ordering and solve the motion planning problem quickly. We evaluate our algorithms on a wide range of 2D and 7D problems and show that the learned selector outperforms baseline commonly used heuristics.**

I. Introduction

In this paper, we explore algorithms that leverage past experience to find the shortest path on a graph while minimizing planning time. We focus on the domain of robot motion planning where the planning time is dominated by *edge evaluation* [1]. Here the goal is to check the minimal number of edges, invalidating potential shortest paths along the way, until we discover the shortest feasible path – this is the central tenet of lazy search [2, 3]. We propose to *learn within this framework* which edges to evaluate (Fig. 1).

How should we leverage experience? Consider the "Piano Mover's Problem" [4] where the goal is to plan a path for a piano from one room in a house to another. Collision checking all possible motions of the piano can be quite time-consuming. Instead, what can we infer if we were given a database of houses and edge evaluations results?

1) *Check doors first* - these edges serve as bottlenecks for many paths which can be eliminated early if invalid.
2) *Prioritize narrow doors* - these edges are more likely to be invalid and can save checking other edges.
3) *Similar doors, similar outcomes* - these edges are correlated, checking one reveals information about others.

Intuitively, we need to consider all past discoveries about edges to make a decision. While this has been explored in the Bayesian setting [5, 6], we show that more generally the problem can be mapped to a Markov Decision Process (MDP). However, the size of the MDP grows exponentially with the size of the graph. Even if we were to use approximate dynamic programming, we still need to explore an inordinate number of states to learn a reasonable policy.

Interestingly, if we were to reveal the status of all the edges during training, we can conceive of a *clairvoyant oracle* [7] that can select the optimal sequence of edges to invalidate. In fact, we show that the oracular selector is equivalent to *set cover*, for which greedy approximations exist. By imitating clairvoyant oracles [7], we can drastically cut down on exploration and focus learning on a small, relevant portion of the state space [8]. This leads to a key insight: use imitation learning to quickly bootstrap the selector to match oracular performance. We propose a new algorithm, STROLL, that deploys an interactive imitation learning framework [9] to train the edge selector (Fig. 2). At every iteration, it samples a world (validity status for all edges) and executes the learner. At every timestep, it queries the clairvoyant oracle associated with the world to select an edge to evaluate. This can be viewed as a classification problem where the goal is to map features extracted from edges to the edge selected by the oracle. This datapoint is aggregated with past data, which is then used to update the learner.

In summary, our main contributions are:

1) We map edge selection in lazy search to an MDP (Section II) and solve it for small graphs (Section III).
2) We show that larger MDPs, can be efficiently solved by imitating clairvoyant oracles (Section IV).
3) We show that the learned policy can outperform competitive baselines on a wide range of datasets (Section V).

II. Problem Formulation

The overall objective is to design an algorithm that can solve the Shortest Path (SP) problem while minimizing the number of edges evaluated.

A. The Shortest Path (SP) Problem

Let $G = (V, E)$ be an explicit graph where V denotes the set of vertices and E the set of edges. Given a start and goal vertex $(v_s, v_g) \in V$, a path ξ is represented as a sequence of vertices $(v_1, v_2, \dots, v_l)$ such that $v_1 = v_s, v_l = v_g, \forall i,\ (v_i, v_{i+1}) \in E$. We define a *world* $\phi : E \to \{0, 1\}$ 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. $\forall e \in \xi, \phi(e) = 1$. Let $\ell : E \to \mathbb{R}^+$ be the length of an edge. The length of a path is the sum of edge lengths, i.e. $\ell(\xi) = \sum_{e \in \xi} \ell(e)$. The objective of the SP problem is the find the shortest feasible path:

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

We now define a family of shortest path algorithms. Given a SP problem, the algorithms evaluate a set of edges $E_{\text{eval}} \subset E$

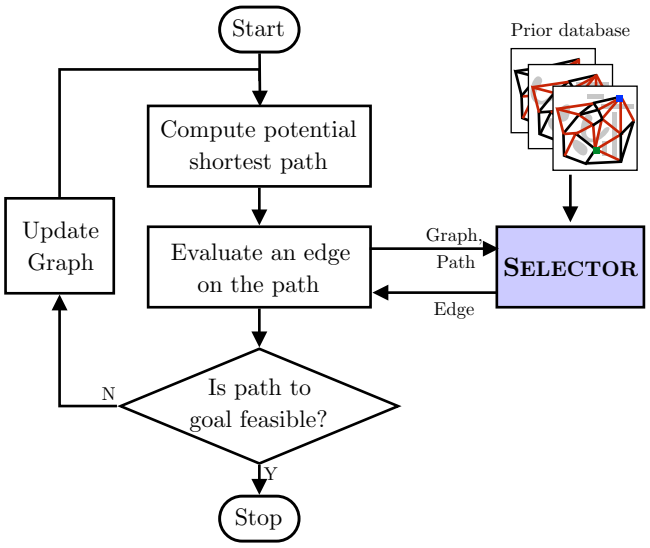


Fig. 1: The LAZYSP [2] framework. LAZYSP iteratively computes the shortest path, queries a SELECTOR for an edge on the path, evaluates it and updates the graph until a feasible path is found. The number of edges evaluated depends on the choice of SELECTOR. We propose to train a SELECTOR from prior data.

(verify if they are valid) and return a path ξ^* upon halting. Two conditions must be met:

1) The returned path ξ^* is verified to be feasible, i.e. $\forall e \in \xi^*,\ e \in E_{\text{eval}},\ \phi(e) = 1$
2) All paths shorter than ξ^* are verified to be infeasible, i.e. $\forall \xi_i,\ \ell(\xi_i) \leq \ell(\xi^*),\ \exists e \in \xi_i,\ e \in E_{\text{eval}},\ \phi(e) = 0$

B. The Lazy Shortest Path (LAZYSP) Framework

We are interested in shortest path algorithms that minimize the number of evaluated edges $|E_{\text{eval}}|$.[1] These are *lazy* algorithms, i.e. they seek to defer the evaluation of an edge as much as possible. When this laziness is taken to the limit, one arrives at the *Lazy Shortest Path* (LAZYSP) class of algorithms. Under a set of assumptions, this framework can be shown to contain the optimally lazy algorithm [10].

Algorithm 1 describes the LAZYSP framework. The algorithm maintains a set of evaluated edges that are valid E_{val} and invalid E_{inv}. At every iteration, the algorithm lazily finds the shortest path ξ on the potentially valid graph $G = (V, E \setminus E_{\text{inv}})$ *without evaluating any new edges* (Line 4). It then calls a function, SELECTOR, to select an edge e from this path ξ (Line 5). Depending on the outcome, this edge is added to either E_{val} or E_{inv}. This process continues until the conditions in Section II-A are satisfied, i.e. the shortest feasible path is found.

The algorithm has one free parameter - the SELECTOR function. The only requirement for a valid SELECTOR is to select an edge on the path. As shown in [2], one can design a range of selectors such as:

1) FORWARD: select the first unevaluated edge $e \in \xi$. Effective if invalid edges are near the start.

[1]The framework can be extended to handle non-uniform evaluation cost as well

Algorithm 1: LAZYSP

Input : Graph G, start v_s, goal v_g, world ϕ
Parameter: SELECTOR
Output : Path ξ^*, evaluated edges E_{eval}

1 $E_{\text{val}} \leftarrow \emptyset$ ▷ Valid evaluated edges
2 $E_{\text{inv}} \leftarrow \emptyset$ ▷ Invaid evaluated edges
3 **repeat**
4 $\quad \xi \leftarrow \text{SHORTESTPATH}(E \setminus E_{\text{inv}})$
5 $\quad e \leftarrow \text{SELECTOR}(\xi, E_{\text{val}}, E_{\text{inv}})$ ▷ Select edge on ξ
6 $\quad$ **if** $\phi(e) \neq 0$ **then**
7 $\quad\quad E_{\text{val}} \leftarrow E_{\text{val}} \cup \{e\}$
8 $\quad$ **else**
9 $\quad\quad E_{\text{inv}} \leftarrow E_{\text{inv}} \cup \{e\}$
10 $\quad$ **end**
11 **until** *feasible path found s.t.* $\forall e \in \xi, e \in E_{\text{val}}$;
12 **return** $\{\xi^* \leftarrow \xi,\ E_{\text{eval}} \leftarrow E_{\text{val}} \cup E_{\text{inv}}\}$;

2) BACKWARD: select the last unevaluated edge $e \in \xi$. Effective if invalid edges are near the goal.
3) ALTERNATE: alternates between first and last edge. This approach hedges its bets between start and goal.
4) FAILFAST: selects the least likely edge $e \in \xi$ to be valid based on prior data.
5) POSTFAILFAST: selects the least likely edge $e \in \xi$ to be valid using a Bayesian posterior based on edges checked so far.

While these baseline selectors are very effective in practice, their performance, i.e. the number of edges evaluated $|E_{\text{eval}}|$ depends on the underlying world ϕ which dictates which edges are invalid. Hence the goal is to compute a good SELECTOR that is effective given a *distribution of worlds*, $P(\phi)$. We formalize this as follows

Problem 1 (Optimal Selector Problem)**.** *Let the edges evaluated by* SELECTOR *on world* ϕ *be denoted by* $E_{\text{eval}}(\phi, \text{SELECTOR})$. *Given a distribution of worlds,* $P(\phi)$, *find a* SELECTOR *that minimizes the expected number of evaluated edges, i.e.* $\min \mathbb{E}_{\phi \sim P(\phi)} \left[|E_{\text{eval}}(\phi, \text{SELECTOR})|\right]$

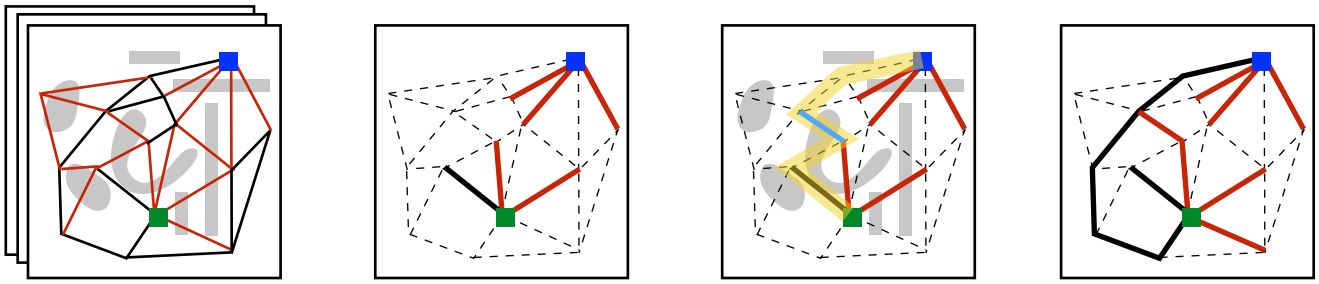
Problem 1 is a sequential decision making problem, i.e. decisions made by the selector in one iteration (edge selected) affects the input to the selector in the next iteration (shortest path). We show how to formally handle this in the next section. It's interesting to note that Problem 1 can be solved optimally under certain strong assumptions (See supplementary for details[2]).

C. Mapping the Optimal Selector Problem to an MDP

We map Problem 1 to a Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, \mathrm{T}, \mathrm{R} \rangle$ as follows:

State Space: The state $s = (E_{\text{val}}, E_{\text{inv}})$ is the set of evaluated valid edges E_{val} and evaluated invalid edges E_{inv}. This can be represented by a vector of size $|E|$, each element

[2]Supplementary material can be found at: http://bit.ly/2Em6Meu



Sample a world from database. | Execute learner (evaluate edges) to a get a state. | Query clairvoyant oracle for edge (action) to evaluate. | Continue aggregating data till the end, update learner.

Fig. 2: Overview of StROLL- a training procedure for a SELECTOR to select edges to evaluate in the LAZYSP framework. In each training iteration, a world map ϕ is sampled. The learner is executed upto a time step to get a state s_t which is the set of edges evaluated and their outcomes. The learner has to decide which edge to evaluate on the current shortest path. It extracts features from every edge - we use a set of baseline heuristic values as features. The SELECTOR asks a clairvoyant oracle selector (which has full knowledge of the world) which edge to evaluate. This is then added to a classification dataset and the learner is updated. This process is repeated over several iterations.

being one of $\{-1, 0, 1\}$ - unevaluated, evaluated invalid, and evaluated valid respectively. For simplicity, we assume that the explicit graph $G = (V, E)$ is fixed.[3]

Since each $e \in E$ can be in one of 3 sets, the cardinality of the state space is $|S| = 3^{|E|}$.

The MDP has an absorbing goal state set $\mathcal{G} \subset \mathcal{S}$ which is a set of states where all the edges on the current shortest path are evaluated to be valid, i.e.

$$\mathcal{G} = \{(E_{\text{val}}, E_{\text{inv}}) \mid \forall e \in \textsc{ShortestPath}(E \setminus E_{\text{inv}}), e \in E_{\text{val}}\} \tag{2}$$

Action Set: The action set $\mathcal{A}(s)$ is the set of unevaluated edges on the current shortest path, i.e.

$$\mathcal{A}(s) = \{e \in \textsc{ShortestPath}(E \setminus E_{\text{inv}}), e \notin \{E_{\text{val}} \cup E_{\text{inv}}\}\} \tag{3}$$

Transition Function: Given a world ϕ, the transition function is deterministic $s' = \Gamma(s, a, \phi)$:

$$\Gamma(s, a, \phi) = \begin{cases} (E_{\text{val}} \cup \{e\}, E_{\text{inv}}) & \text{if } \phi(e) = 1 \\ (E_{\text{val}}, E_{\text{inv}} \cup \{e\}) & \text{if } \phi(e) = 0 \end{cases} \tag{4}$$

Since ϕ is latent and distributed according to $P(\phi)$, we have a stochastic transition function $\mathrm{T}(s, a, s') = \sum_\phi P(\phi)\mathbb{I}(s = \Gamma(s, a, \phi))$.

Reward Function: The reward function penalizes every state other than the absorbing goal state $\mathcal{G}$, i.e.

$$\mathrm{R}(s, a) = \begin{cases} 0 & \text{if } s \in \mathcal{G} \\ -1 & \text{otherwise} \end{cases} \tag{5}$$

III. CHALLENGES IN SOLVING THE MDP

In this section, we examine tiny graphs and show that even for such problems, a choice of world distributions where edges are correlated can affect SELECTOR choices. However, by solving the MDP using tabular Q-learning we can automatically recover the optimal SELECTOR.

A. Experimental setup

We train selectors on two different graphs and corresponding distribution of worlds $P(\phi)$.

[3]We can handle a varying graph by adding it to the state space.

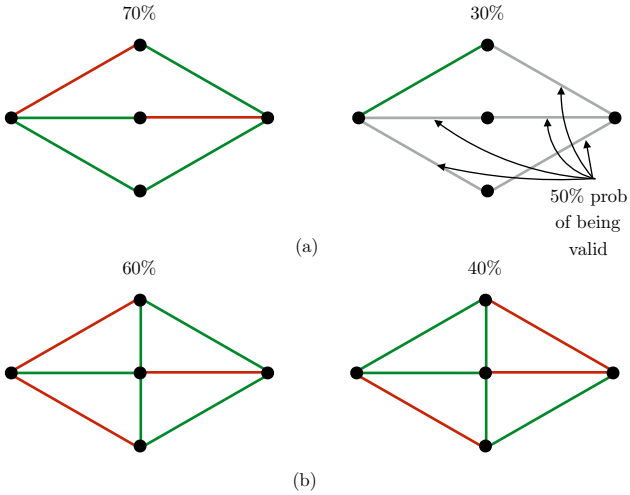


Fig. 3: Distribution over worlds for (a) Environment 1 and (b) Environment 2. The goal is to find a path from left to right. Edges are valid (green) or invalid (red)

Environment 1: Fig. 3(a) illustrates the distribution of Environment 1. The graph has 6 edges. With 70% probability, `top_left` edge is invalid. If `top_left` is invalid, then `middle_right` is always invalid. If `top_left` is valid, then with 50% probability, `top_right` is invalid plus any one of remaining four are invalid.

The optimal policy is to check `top_left` edge first.

- *If invalid*, check `middle_right` (which is necessarily invalid) and check bottom two edges which are feasible. This amounts to 4 evaluated edges.
- *If valid*, check other edges in order as they all have 50% probability of being valid.

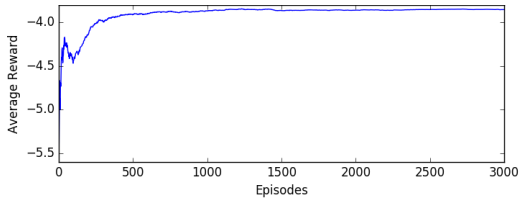
Environment 2: Fig. 3(b) illustrates the distribution of Environment 2. The graph has 8 edges. With 60% of the time `top_left`, `middle_right` and `bottom_left` are invalid. Else, `top_right` and `middle_right` are invalid. Intuitively, 60% of the time, SELECTALTERNATE is optimal and 40% of the time, SELECTBACKWARD is the best.

TABLE I: Q-learning parameters.

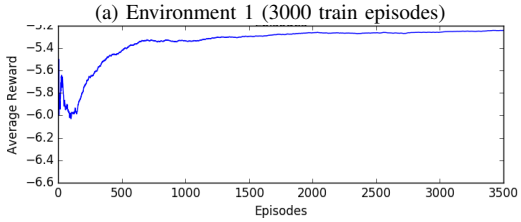
Parameter	Environment 1	Environment 2
Number of episodes	3000	3500
Exploration episodes	100	150
ϵ_0	1	1
Discount factor	1	1
Learning rate	0.5	0.5

TABLE II: Average reward after 1000 test episodes.

Method	Environment 1	Environment 2
Tabular Q-learning	-3.85	-5.24
FORWARD	-4.54	-6.00
BACKWARD	-4.42	-5.79
ALTERNATE	-3.86	-6.00
RANDOM	-4.48	-5.90



(a) Environment 1 (3000 train episodes)



(b) Environment 2 (3500 train episodes)

Fig. 4: Average reward per epsiode of Tabular Q-learning.

B. Solving the MDP via Q-learning

We apply tabular Q-learning [11] to compute the optimal value $Q^*(s,a)$. Broadly speaking, the algorithm uses an $\epsilon-$greedy algorithm to visit states, gather rewards, and perform Bellman backups to update the value function. Environment 1 has 729 states, Environment 2 has 6561 states. The learning parameters are shown in Table I.

Fig. 4 shows the average reward during training for Q-learning. Environment 1 converges after $\approx$ 1000 episodes, environment 2 after ≈ 3000 episodes. Table II shows a comparison of Q-learning with other heuristic baselines in terms of average reward on a validation dataset of 1000 problems. In Environment 1, the learner discovers the optimal policy. Interestingly, ALTERNATE also achieves this result since the correlated edges are alternating. In Environment 2, the learner has a clear margin as compared to heuristic baselines, all of which are vulnerable to one of the modes.

This shows that, even on such small graphs, it is possible to create an environment where heuristic baselines fail. The fact that the learner can recover optimal policies is promising.

C. Challenges on scaling to larger graphs

While we can solve the MDP for tiny graphs, we run into a number of problems as we try to scale to larger graphs:

a) Exponentially large state space: The size of the state space is $|S| = 3^{|E|}$. This leads to exponentially slower convergence rates as the size of the graph increases. Even if we could manage to visit only the relevant portion of this space, this approach would not generalize across graphs.

b) Convergence issues with approximate value iteration: We can scale to large graphs if we use a function approximator. In this case, we have to featurize (s,a) as a vector f, i.e. we are trying to approximate $Q(s,a) \approx Q(f)$. Fortunately, we have a set of baseline heuristics II-B that can be used as a feature vector. This choice allows us to potentially improve upon baselines and easily switch between problem domains.

We run into another problem - approximate value iteration is not guaranteed to converge [12]. This is exaggerated in our case where f is a set of baseline heuristics that may not retain the same information content as the state s. Hence multiple states map to the same feature f, which leads to oscillations and local minima.

c) Sparse rewards: Every state gets a penalization except the absorbing state, i.e. rewards are sparse. Because we are using a function approximator, updates to $Q(f)$ for reaching the goal state are overridden by updates due to -1 penalization.

IV. APPROACH

Our approach, STROLL (**S**earch **thr**ough **O**racle **L**earning and **L**aziness), is to imitate clairvoyant oracles that can show how to evaluate edges optimally given full knowledge of the MDP at training time. To deal with distribution mismatch between oracle and learner, we use established techniques for iterative supervised learning.

A. Optimistic Value Estimate using a Clairvoyant Oracle

Consider the situation where the world ϕ is fully known to the selector, i.e. the $0/1$ status of all edges are known. The selector can then judiciously select edges that are not only invalid, but eliminate paths quickly. We call such a selector a *clairvoyant oracle*. We show that the optimal clairvoyant oracle, that evaluates the minimal number of edges, is the solution to a set cover problem.

Theorem 1 (Clairvoyant Oracle as Set Cover)**.** *Let $s = (E_{\text{val}}, E_{\text{inv}})$ be a state. Let $V^*(s,\phi)$ be the optimal state action value when the world ϕ is known. Then $V^*(s,\phi)$ is the solution to the following set cover problem*

$$-\min_{E_{\text{eval}} \subset \{e \in E \mid \phi(e)=0\}} |E_{\text{eval}}| \quad \text{s.t.} \quad \forall \xi,\ \ell(\xi) \le \ell(\xi^*), \xi \cap E_{\text{inv}} = \emptyset,\ \xi \cap E_{\text{eval}} \neq \emptyset \tag{6}$$

where ξ^* is the shortest feasible path for world ϕ.

Proof: (Sketch) Let $\Xi = \{\xi_1, \ldots, \xi_n\}$ be the set of paths that satisfy the constraints of (6)

1) Shorter than ξ^*, i.e. $\ell(\xi_i) \le \ell(\xi^*)$
2) Paths are not yet invalidated i.e. $\xi \cap E_{\text{inv}} = \emptyset$

Let $\{e \in E \mid \phi(e) = 0\}$ be the set of invalid edges. Each edge e covers a path $\xi_i \in \Xi$ if $e \in \xi_i$. We define a cover as a set of edges E_{eval} that covers all paths in Ξ, i.e. $\xi_i \cap E_{\text{eval}} \neq \emptyset$.

Algorithm 2: APPROXIMATE CLAIRVOYANT ORACLE

Input : State $s = (E_{\text{val}}, E_{\text{inv}})$, world ϕ
Output: Action a

Compute shortest path $\hat{\xi} = \text{SHORTESTPATH}(E \setminus E_{\text{inv}})$
$\Delta \leftarrow 0_{|E|\times 1}$
for $e \in \hat{\xi}, \phi(e) = 0$ **do**
  $\Delta(e) \leftarrow \ell(\text{SHORTESTPATH}(E \setminus \{E_{\text{inv}} \cup \{e\}\})) - \ell(\hat{\xi})$
return Action $a = \arg\max_{e \in \hat{\xi}} \Delta(e)$;

If we select a min cover, i.e. $\min |E_{\text{eval}}|$ then all shorter paths will be eliminated. Hence this is equal to the optimal value $-V^*(s, \phi)$. ∎

Theorem 1 says that given a world and a state of the search, the clairvoyant oracle selects the minimum set of invalid edges to eliminate paths shorter than the shortest feasible path.

Let $\pi_{\text{OR}}(s, \phi)$ be the corresponding oracle policy. We note that the optimal clairvoyant oracle can be used to derive an upper bound for the optimal value

$$Q^*(s, a) \leq Q^{\pi_{\text{OR}}}(s, a) = \sum_{\phi} P(\phi|s) Q^{\pi_{\text{OR}}}(s, a, \phi) \tag{7}$$

where $P(\phi|s)$ is the posterior distribution over worlds given state and $Q^{\pi_{\text{OR}}}(s, a, \phi)$ is the value of executing action a in state s and subsequently rolling-out the oracle. Hence this upper bound can be used for learning.

B. Approximating the Clairvoyant Oracle

Since set cover is NP-Hard, we have to approximately solve (6). Fortunately, a greedy approximation exists which is near-optimal. The greedy algorithm iterates over the following rule:

$$\begin{aligned} e_i &= \arg\max_{e \in E, \phi(e)=0} |\{\xi \mid \ell(\xi) \leq \ell(\xi^*),\ \xi \cap E_{\text{inv}} = \emptyset,\ e \in \xi\}| \\ E_{\text{eval}} &\leftarrow E_{\text{eval}} \cup \{e_i\} \end{aligned} \tag{8}$$

The approach greedily selects an invalid edge that covers the maximum number of shorter paths, which have not yet been eliminated. This greedy process is repeated until all paths are eliminated.

There are two practical problems with computing such an oracle. First, enumerating all shorter paths $\{\xi \mid \ell(\xi) \leq \ell(\xi^*)\}$ is expensive, even at train time. Second, if we simply wish to query the oracle for which edge to select on the current shortest path $\hat{\xi} = \text{SHORTESTPATH}(E \setminus E_{\text{inv}})$, it has to execute (8) potentially multiple times before such an edge is discovered - which also can be expensive. Hence we perform a double approximation.

The first approximation to (8) is to constrain the oracle to only select an edge on the current shortest path $\hat{\xi} = \text{SHORTESTPATH}(E \setminus E_{\text{inv}})$

$$\approx \arg\max_{e \in \hat{\xi},\ \phi(e)=0} |\{\xi \mid \ell(\xi) \leq \ell(\xi^*),\ \xi \cap E_{\text{inv}} = \emptyset,\ e \in \xi\}| \tag{9}$$

The second approximation to (9) is to replace the number of paths covered with the marginal gain in path length on invalidating an edge.

$$\approx \arg\max_{e \in \hat{\xi},\ \phi(e)=0} \ell(\text{SHORTESTPATH}(E \setminus \{E_{\text{inv}} \cup \{e\}\})) - \ell(\hat{\xi}) \tag{10}$$

Alg. 2 summarizes this approximate clairvoyant oracle.

C. Bootstrapping with Imitation Learning

Imitation learning is a principled way to use the clairvoyant oracle $\pi_{\text{OR}}(s, \phi)$ to assist in training the learner $\pi(s)$. In our case, we can use the oracle action value $Q^{\pi_{\text{OR}}}(s, a)$ as a target for our learner as follows:

$$\arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi(s)} \left[Q^{\pi_{\text{OR}}}(s, \pi(s))\right] \tag{11}$$

where $d_\pi(s)$ is the distribution of states. Note that this is now a classification problem since the labels are provided by the oracle. However the distribution d_π depends on the learner's π. Ross and Bagnell [13] show that this type of imitation learning problem can be reduced to interactive supervised learning.

We simplify further. Computing the oracle value requires rolling out the oracle until termination. We empirically found this to significantly slow down training time. Instead, we train the policy to directly predict the action that is selected by the oracle. This is the same as (11) but with a $0/1$ loss [9] -

$$\arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi(s)} \left[\mathbb{I}(\pi(s) = \pi_{\text{OR}}(s, \phi))\right] \tag{12}$$

We justify this simplification by first showing that maximizing action value is same as maximizing the advantage $Q^{\pi_{\text{OR}}}(s, a) - V^{\pi_{\text{OR}}}(s)$. Since all the rewards are -1, the advantage can be lower bounded by the $0/1$ loss. We summarize this as follows:

$$\begin{aligned} & \max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi(s)} \left[Q^{\pi_{\text{OR}}}(s, \pi(s))\right] \\ = & \max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi(s)} \left[Q^{\pi_{\text{OR}}}(s, \pi(s)) - V^{\pi_{\text{OR}}}(s)\right] \\ \geq & \max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi(s)} \left[\mathbb{I}(\pi(s) = \pi_{\text{OR}}(s, \phi)) - 1\right] \end{aligned} \tag{13}$$

Finally, we do not use the exact clairvoyant oracle but rather an approximation (Section IV-B). In other words, there can exist policies $\pi \in \Pi$ that outperform the oracle. In such a case, one can potentially apply policy improvement after imitation learning. However, we leave the exploration of this direction to future work.

D. Algorithm

The problem in (12) is a non-*i.i.d* classification problem - the goal is to select the same action the oracle would select on the *on policy distribution of learner*. Ross et al. [9] proposed an algorithm, DAGGER, to exactly solve such problems.

Alg. 3, describes the STROLL framework which iteratively trains a sequence of policies $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$. At every iteration i, we collect a dataset $\mathcal{D}_i$ by executing m different episodes. In every episode, we sample a world ϕ which already has every edge evaluated. We then roll-in a policy (execute a selector) which is a mixture π_{mix} that blends the learner's current policy, $\hat{\pi}_i$ and a base roll-in policy π_{roll} using blending parameter β_i. At every time step t, we query the clairvoyant

Algorithm 3: StROLL

Input : World distribution $P(\phi)$, oracle π_{OR}
Parameter: Iter N, roll-in policy π_{roll}, mixing $\{\beta_i\}_{i=1}^{N}$
Output : Policy $\hat{\pi}$

Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π
for $i = 1, \ldots, N$ **do**
  Initialize sub-dataset $\mathcal{D}_i \leftarrow \emptyset$
  Let mixture policy be $\pi_{\text{mix}} = \beta_i \pi_{\text{roll}} + (1 - \beta_i)\hat{\pi}_i$
  for $j = 1, \ldots, m$ **do**
    Sample $\phi \sim P(\phi)$;
    Rollin π_{mix} to get state trajectory $\{s_t\}_{t=1}^{T}$
    Invoke oracle to get $a_t = \pi_{\text{OR}}(s_t, \phi)$
    $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{(s_t, a_t)\}_{t=1}^{T}$;
  Aggregate data $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$;
  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$;
return Best $\hat{\pi}$ on validation;

oracle with state s_t to receive an action a_t. We use the approximate oracle in Alg. 2. We then extract a feature vector f from all (s_t, a) tuples and create a classification datapoint. We add this datapoint to the dataset $\mathcal{D}_i$. At the end of m episodes, this data is then *aggregated with the existing dataset* $\mathcal{D}$. A new classifier $\hat{\pi}_{i+1}$ is trained on the aggregated data. At the end of N iterations, the algorithm returns the best performing policy on a set of held-out validation environments.

We have two algorithms based on the choice of π_{roll}:

1) StROLL: We set $\pi_{\text{roll}} = \pi_{\text{OR}}$. This is the default mode of DAGGER. This uses the oracle state distribution to stabilize learning initially.
2) StROLL-R: We set π_{roll} to be the best performing heuristic on training as defined in Section II-B. This uses a heuristic state distribution to stabilize learning. Since the heuristic is realizable, it can have a stabilizing effect on datasets where the oracle is far from realizable.

We inherit the performance guarantees of DAGGER [9], which bounds the performance gap with respect to the best policy in the policy class.

V. Experiments

A. Experimental Setup

We use datasets from [5] in our experiments. The 2D datasets contain graphs with approximately 1600-5000 edges and varied obstacle distributions. The two 7D datasets involve a robot arm planning for a reaching task in clutter with large graphs containing 33286 edges.

Learning Details: We only consider policies that are a linear combination of a minimal set of features, where each feature is a different motion planning heuristic. The features we consider are:

1) PRIOR- the prior probability of an edge being invalid calculated over the training dataset.
2) POSTERIOR- the posterior probability of an edge being invalid given collision checks done thus far (See supplementary for details).
3) LOCATION- score ranging from 1 (first unchecked edge) to 0 (last unchecked edge).
4) Δ-LENGTH- hallucinate that an edge is invalid, then calculate the difference in length of new shortest path compared with the current shortest path.
5) Δ-EVAL- hallucinate that an edge is invalid, the calculate the fraction of unevaluated edges on the new shortest path.
6) PΔ-LENGTH- calculated as POSTERIOR $\times$ Δ-LENGTH, it weighs the Δ-LENGTH of an edge with the probability of it being invalid and is effective in practice (Table III).

B. Baselines

We compare our approach to common heuristics used in LAZYSP as described in Section II-B. We also analyze the improvement in performance as compared to vanilla behavior cloning of the oracle and reinforcement learning from scratch.

C. Analysis of Overall Performance

O 1. StROLL *has consistently strong performance across different datasets.*

Table III shows that StROLL is able to learn policies competitive with other motion planning heuristics. No other heuristic has as consistent a performance across datasets.

O 2. *The learner focuses collision checking on edges that are highly likely to be invalid and have a high measure of centrality.*

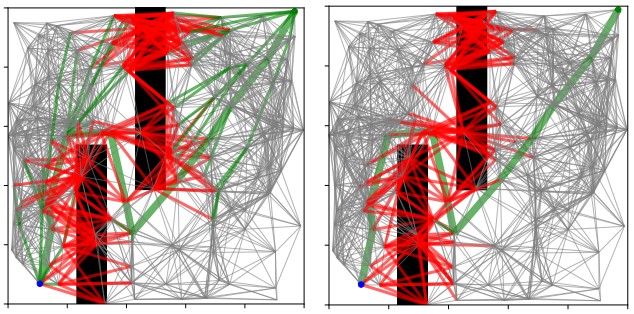
Fig. 8 shows the activation of different features across datasets. The learner places high importance on POSTERIOR, Δ-LENGTH and PΔ-LENGTH. POSTERIOR is an approximate likelihood of an edge being invalid and Δ-LENGTH is an approximate measure of centrality i.e. edges with large Δ-LENGTH have large number of paths passing through them (Note that the converse may not always apply).

O 3. *On datasets with strong correlations among edges, heuristics that take obstacle distribution into account outperform uninformed heuristics, and* StROLL *is able to learn significantly better policies than uninformed heuristics.*

Examples of such datasets are GATE, BAFFLE, BUGTRAP and BLOB. Here, StROLL and StROLL-R eliminate a large number of paths by only evaluating edges which are highly likely to be in collision and have several paths passing through them (Fig. 5). In the 7D datasets, obstacles are highly concentrated near the goal region, which explains the strong performance of the uninformed BACKWARD selector. However, due to a very large number of edges and limited training sets, POSTERIOR and Δ-LENGTH are inaccurate causing the learner to fail to outperform BACKWARD.

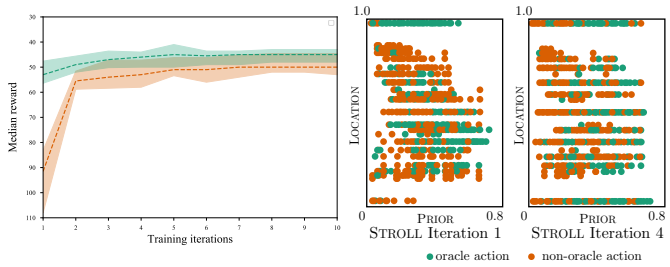
O 4. *On datasets with uniformly spread obstacles, uninformed heuristics can perform better than* StROLL.

Examples of such datasets are TWOWALL and FOREST where the lack of structures makes features such as posterior



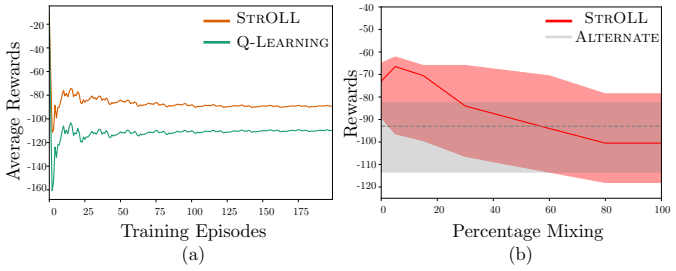
(a) ALTERNATE $|E_{\text{eval}}| = 292$ (b) STROLL $|E_{\text{eval}}| = 180$

Fig. 5: Edges evaluated (green valid, red invalid) on an environment from ONEWALL. (a) ALTERNATE evaluates several valid edges (b) STROLL evaluates many fewer edges, all of which are invalid and eliminate a large number of paths.



(a) (b)

Fig. 6: (a) STROLL-R (green) vs STROLL (orange) (b) Densification of data.



(a) (b)

Fig. 7: (a) Running average reward for 200 episodes of training. Q learning suffers due to large state space and sparse rewards. (b) Performance on validation set of 200 worlds with contamination from different distribution.

uninformative. This combined with the non-realizability of the oracle makes it difficult for STROLL to learn a strong policy.

D. Case Studies

Q 1. *How does performance vary with training data?*

Fig. 6(a) shows the improvement in median validation reward with an increasing number of training iterations. Also, Fig. 6(b) shows that with more iterations, the learner visits diverse parts of the state-space on x-axis not visited by the oracle.

Q 2. *How significant is the impact of heuristic roll-in on stabilizing learning in high-dimensional problems?*

Fig. 6 shows a comparison of the median validation return per iteration using STROLL versus STROLL-R on CLUTTER1 and CLUTTER2 datasets. Heuristic roll-in helps converge to a better policy in lesser number of iterations. Interestingly, the policy learned in the first iteration of STROLL-R is significantly better than STROLL, demonstrating the stabilizing effects of heuristic roll-in.

Q 3. *How does performance compare to reinforcement learning with function approximation?*

Fig. 7(a) shows training curves for STROLL and Q-LEARNING with linear function approximation and experience replay. STROLL is more sample efficient and converges to a competitive policy faster.

Q 4. *How does performance vary with train-test mismatch?*

Fig. 7(b) shows a stress-test of a policy learned on ONE WALL by running it on a validation set which is increasingly contaminated by environments from FOREST. The learned policy performs better than the best uninformed heuristic on FOREST for up to 60% contamination.

VI. RELATED WORK

In domains where edge evaluations are expensive and dominate planning time, a *lazy approach* is often employed [3] wherein the graph is constructed *without* testing if edges are collision-free. LAZYSP [2] extends the graph all the way to the goal, before evaluating edges. LWA* [14] extends the graph only a single step before evaluation. (LRA*) [15] is able to trade-off between them by allowing the search to go to an arbitrary lookahead. The principle of laziness is reflected in similar techniques for randomized search [1, 16].

Several previous works investigated leveraging priors in search. FuzzyPRM [17] evaluates paths that minimize the probability of collision. The Anytime Edge Evaluation (AEE*) framework [18] uses an anytime strategy for edge evaluation informed by priors. BISECT [5] and DIRECT [6] casts search as Bayesian active learning to derive edge evaluation. However, these methods make specific assumptions about the graph or about the priors. Our approach is more general.

Efficient collision checking has its own history in the context of motion planning. Other approaches model belief over the configuration space to speed-up collision checking [19, 20], sample vertices in promising regions [21] or grow the search tree to explore the configuration space [22–24]. However, these approaches make geometric assumptions and rely on domain knowledge. We work directly with graphs and are agnostic with respect to the domain.

Several recent works use imitation learning [8, 9, 13] to bootstrap reinforcement learning. THOR [25] performs a multi-step search to gain advantage over the reference policy. LOKI [26] switches from IL to RL. Imitation of clairvoyant oracles has been used in multiple domains like information gathering [7], heuristic search [27], and MPC [28, 29].

VII. DISCUSSION

We examined the problem of minimizing edge evaluations in lazy search on a distribution of worlds. We first formulated the problem of deciding which edge to evaluate as an MDP and presented an algorithm to learn policies by imitating clairvoyant oracles, which, if the world is known, can optimally evaluate edges. While imitation learning of clairvoyant oracles is effective, the approach may be further improved through

TABLE III: Edges evaluated by different algorithms across different datasets (median, upper and lower C.I on 200 held-out environments). Highlighted is the best performing selector in terms of median score not counting the oracle.

	ORACLE	BACKWARD	ALTERNATE	FAILFAST	POSTFAILFAST	PΔLENGTH	SUPERVISED	STROLL	STROLLR
2D Geometric Planning									
ONEWALL	$80.0^{+6.0}_{-48.0}$	$87.0^{+8.4}_{-41}$	$112.0^{+12.8}_{-60.0}$	$82.0^{+3.0}_{-47.0}$	$81.0^{+3.0}_{-49.0}$	$85.0^{+6.8}_{-52.6.0}$	$79.0^{+3.0}_{-45.0}$	$79.0^{+5.0}_{-44.8}$	$79.0^{+5.0}_{-44.8}$
TWOWALL	$107.0^{+23.0}_{-0.0}$	$199.0^{+8.0}_{-19.0}$	$138.0^{+7.0}_{-2.0}$	$178.0^{+0.0}_{-6.0}$	$177.0^{+0.0}_{-7.0}$	$120.0^{+19.0}_{-0.0}$	$177.0^{+0.0}_{-6.0}$	$177.0^{+0.0}_{-6.0}$	$170.0^{+12.2}_{-0.0}$
FOREST	$90^{+14.4}_{-10.0}$	$128.0^{+15.0}_{-16.4}$	$115.0^{+12.0}_{-13.2}$	$135.0^{+13.0}_{-16.0}$	$116.0^{+13.2}_{-16.4}$	$102.0^{+15.0}_{-12.0}$	$117.0^{+19.2}_{-17.0}$	$115.0^{+20.0}_{-15.0}$	$115.0^{+21.2}_{-13.4}$
GATE	$50.0^{+6.0}_{-9.0}$	$74.0^{+8.0}_{-9.0}$	$75.0^{+14.2}_{-6.2}$	$60.0^{+8.0}_{-6.2}$	$50.0^{+7.0}_{-8.2}$	$53.0^{+7.0}_{-9.2}$	$50.0^{+7.0}_{-7.2}$	$48.0^{+10.0}_{-7.2}$	$48.0^{+9.2}_{-9.2}$
MAZE	$537.0^{+37.0}_{-24.6}$	$668.5^{+40.3}_{-56.1}$	$613.0^{+39.6}_{-33}$	$512^{+52.2}_{-34.0}$	$516.5^{+33.70}_{-36.50}$	$529.0^{+40.0}_{-37.2}$	$502.5^{+58.7}_{-28.2}$	$512.0^{+42.0}_{-31.0}$	$554.0^{+52.2}_{-59.0}$
BAFFLE	$219.0^{+18.0}_{-12.0}$	$244.0^{+14.6}_{-6.0}$	$311.0^{+8.0}_{-15.6}$	$232.0^{+6.0}_{-12.0}$	$211.0^{+7.8}_{-6.0}$	$230.0^{+18.0}_{-17.0}$	$206.0^{+6.8}_{-3.0}$	$205.0^{+6.0}_{-3.0}$	$207.0^{+7.0}_{-2.0}$
BUGTRAP	$77.0^{+12.0}_{-9.4}$	$104.0^{+6.0}_{-14.0}$	$112.5^{+16.9}_{-11.5}$	$90.5^{+10.9}_{-13.5}$	$75.5^{+16.5}_{-6.5}$	$84.5^{+12.9}_{-10.5}$	$75.0^{+15.4}_{-6.4}$	$75.0^{+15.4}_{-6.4}$	$75.0^{+15.4}_{-6.4}$
BLOB	$72.0^{+12.0}_{-4.0}$	$92.0^{+5.4}_{-3.4}$	$109.0^{+5.0}_{-7.0}$	$70.0^{+6.0}_{-6.0}$	$70.0^{+6.0}_{-6.0}$	$80.0^{+9.0}_{-3.0}$	$72.0^{+5.8}_{8.0}$	$70.0^{+6.0}_{-6.0}$	$70.0^{+6.0}_{-6.0}$
7D Manipulation Planning									
CLUTTER1	$35.5^{+1.5}_{-1.5}$	$38.0^{+12.0}_{-10.0}$	$44.0^{+14.2}_{-4.0}$	$92.0^{+5.0}_{-0.0}$	$88.0^{+9.6}_{-0.0}$	$37.5^{+2.1}_{-1.5}$	$95.5^{+17.7}_{-11.1}$	$50.0^{+3.0}_{-6.0}$	$45.0^{+2.0}_{-1.6}$
CLUTTER2	$34.0^{+1.0}_{-2.0}$	$32.0^{+3.0}_{-4.0}$	$41.0^{+2.0}_{-2.2}$	$85.0^{+0.0}_{-3.0}$	$84.0^{+0.0}_{-2.0}$	$37.0^{+0.0}_{-5.0}$	$104.0^{+6.2}_{-6.8}$	$47.0^{+2.0}_{-11.2}$	$44.0^{+5.0}_{-7.2}$

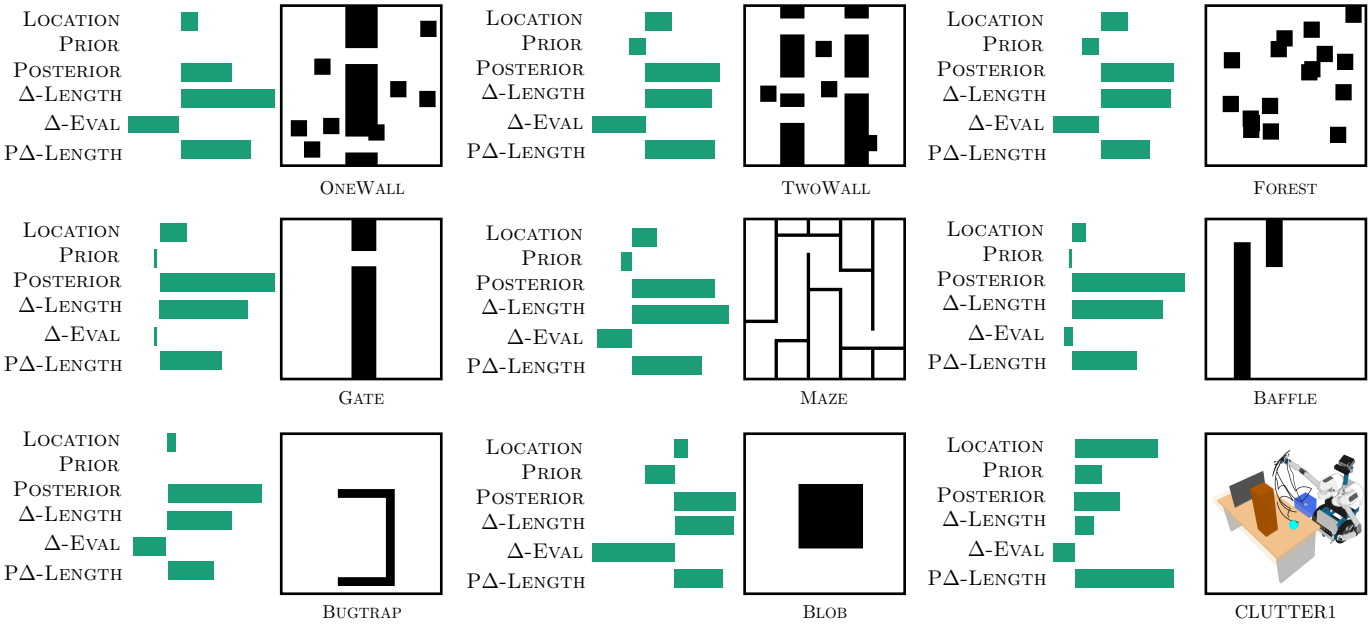


Fig. 8: Weight bins depicting relative importance of each feature learned by the learner. STROLL focuses on edges that are highly likely to be invalid and have high measure of centrality.

reinforcement learning [25, 26]. There are two arguments for this. First, in practice we do not use the exact oracle but a sub-optimal approximation. Second, even if we could use the exact oracle, it may not be realizable by the policy.

ACKNOWLEDGMENTS

This work was (partially) funded by the National Institute of Health R01 (#R01EB019335), National Science Foundation CPS (#1544797), National Science Foundation NRI (#1637748), National Science Foundation CAREER (#1750483), the Office of Naval Research, the RCTA, Amazon, and Honda Research Institute USA.

REFERENCES

[1] Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *ICRA*, 2015.

[2] Christopher M Dellin and Siddhartha S Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *ICAPS*, 2016.

[3] Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *ICRA*, 2000.

[4] Jacob T Schwartz and Micha Sharir. On the "piano movers'" problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on pure and applied mathematics*, 36(3): 345–398, 1983.

[5] Sanjiban Choudhury, Shervin JAvdani, Siddhartha Srinivasa, and Sebastian Scherer. Near-optimal edge evaluation in explicit generalized binomial graphs. In *NIPS*, 2017.

[6] S. Choudhury, S.S. Srinivasa, and S. Scherer. Bayesian active edge evaluation on expensive graphs. In *IJCAI*, 2018.

[7] Sanjiban Choudhury, Mohak Bhardwaj, Sankalp Arora, Ashish Kapoor, Gireeja Ranade, Sebastian Scherer, and Debadeepta Dey. Data-driven planning via imitation learning. *IJRR*, 2017.

[8] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *International Conference on Machine Learning*, pages 3309–3318, 2017.

[9] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, page 6, 2011.

[10] Nika Haghtalab, Simon Mackenzie, Ariel D. Procaccia, Oren Salzman, and Siddhartha S. Srinivasa. The Provable Virtue of Laziness in Motion Planning. pages 106–113, 2018. URL https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17726.

[11] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[12] Geoffrey J Gordon. Stable function approximation in dynamic programming. In *Machine Learning Proceedings 1995*, pages 261–268. Elsevier, 1995.

[13] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv*, 2014.

[14] Benjamin Cohen, Mike Phillips, and Maxim Likhachev. Planning single-arm manipulations with n-arm robots. In *Eigth Annual Symposium on Combinatorial Search*, 2015.

[15] Aditya Mandalika, Oren Salzman, and Siddhartha Srinivasa. Lazy Receding Horizon A* for Efficient Path Planning in Graphs with Expensive-to-Evaluate Edges. pages 476–484, 2018.

[16] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Batch Informed Trees: Sampling-based optimal planning via heuristically guided search of random geometric graphs. In *ICRA*, 2015.

[17] Christian L Nielsen and Lydia E Kavraki. A 2 level fuzzy prm for manipulation planning. In *IROS*, 2000.

[18] Venkatraman Narayanan and Maxim Likhachev. Heuristic search on graphs with existence priors for expensive-to-evaluate edges. In *ICAPS*, 2017.

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

[20] Shushman Choudhury, Christopher M Dellin, and Siddhartha S Srinivasa. Pareto-optimal search over configuration space beliefs for anytime motion planning. In *IROS*, 2016.

[21] Joshua Bialkowski, Michael Otte, and Emilio Frazzoli. Free-configuration biased sampling for motion planning. In *IROS*, 2013.

[22] David Hsu, J-C Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *ICRA*, 1997.

[23] Brendan Burns and Oliver Brock. Sampling-based motion planning using predictive models. In *ICRA*, 2005.

[24] Bakir Lacevic, Dinko Osmankovic, and Adnan Ademovic. Burs of free c-space: a novel structure for path planning. In *ICRA*. IEEE, 2016.

[25] Wen Sun, J Andrew Bagnell, and Byron Boots. Truncated horizon policy search: Combining reinforcement learning & imitation learning. *arXiv preprint arXiv:1805.11240*, 2018.

[26] Ching-An Cheng, Xinyan Yan, Nolan Wagener, and Byron Boots. Fast policy learning through imitation and reinforcement. *arXiv preprint arXiv:1805.10413*, 2018.

[27] Mohak Bhardwaj, Sanjiban Choudhury, and Sebastian Scherer. Learning heuristic search via imitation. In *CoRL*, 2017.

[28] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Plato: Policy learning using adaptive trajectory optimization. In *ICRA*, 2017.

[29] Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Learning from the hindsight plan–episodic mpc improvement. *arXiv preprint arXiv:1609.09001*, 2016.