Learning Heuristic Search via Imitation

Mohak Bhardwaj
The Robotics Institute
Carnegie Mellon University
mohakbhardwaj@cmu.edu

Sanjiban Choudhury
The Robotics Institute
Carnegie Mellon University
sanjiban@cmu.edu

Sebastian Scherer
The Robotics Institute
Carnegie Mellon University
basti@cs.cmu.edu

Abstract: Robotic motion planning problems are typically solved by constructing a search tree of valid maneuvers from a start to a goal configuration. Limited onboard computation and real-time planning constraints impose a limit on how large this search tree can grow. Heuristics play a crucial role in such situations by guiding the search towards potentially good directions and consequently minimizing search effort. Moreover, it must infer such directions in an efficient manner using only the information uncovered by the search up until that time. However, state of the art methods do not address the problem of computing a heuristic that *explicitly minimizes search effort*. In this paper, we do so by training a heuristic policy that maps the partial information from the search to decide which node of the search tree to expand. Unfortunately, naively training such policies leads to slow convergence and poor local minima. We present SAIL, an efficient algorithm that trains heuristic policies by imitating *clairvoyant oracles* - oracles that have full information about the world and demonstrate decisions that minimize search effort. We leverage the fact that such oracles can be efficiently computed using dynamic programming and derive performance guarantees for the learnt heuristic. We validate the approach on a spectrum of environments which show that SAIL consistently outperforms state of the art algorithms. Our approach paves the way forward for learning heuristics that demonstrate an anytime nature - finding feasible solutions quickly and incrementally refining it over time. Open-source code and details can be found here: https://goo.gl/YXkQAC

1 Introduction

Search based motion planning offers a comprehensive framework for reasoning about a vast number of motion planning algorithms [1]. In this framework, an algorithm grows a *search tree* of feasible robot motions from a start configuration towards a goal [2]. This is done in an incremental fashion by first selecting a leaf node of the tree, *expanding* this node by computing outgoing edges, checking each edge for validity and finally updating the tree with potentially new leaf nodes. It is useful to visualize this search process as a *wavefront of expanded nodes* that grows from the start outwards till it finds the goal as illustrated in Fig. 1.

This paper addresses a class of robotic motion planning problems where edge evaluation dominates the search effort, such as for robots with complex geometries like robot arms [3] or for robots with limited onboard computation like UAVs [4]. In order to ensure real-time performance, algorithms must prioritize minimizing the search effort, i.e. keeping the volume of the search wavefront as small as possible while it grows towards the goal. This is typically achieved by heuristics, which guide the search towards promising areas by selecting which nodes to expand. As shown in Fig. 1, this acts as a force stretching the search wavefront towards the goal.

A good heuristic must balance the bi-objective criteria of finding a good solution and minimizing the search effort. The bulk of the prior work has focussed on the former objective of guaranteeing that the search returns a near-optimal solution [2]. These approaches define a heuristic function as a *distance metric* that estimates the cost-to-go value of a node [5]. However, estimation of this distance metric is difficult as it's a complex function of robot geometry, dynamics and obstacle configuration. Commonly used heuristics such as the euclidean distance do not adapt to different robot configurations or different environments. On the other hand, by trying to compute a more accurate distance the heuristic should not end up doing more computation than the original search. While state of the art methods propose different relaxation-based [6, 7] and learning-based approaches [8] to estimate the distance metric they run into a much more fundamental limitation - *a small estimation error can*

1st Conference on Robot Learning (CoRL 2017), Mountain View, United States.

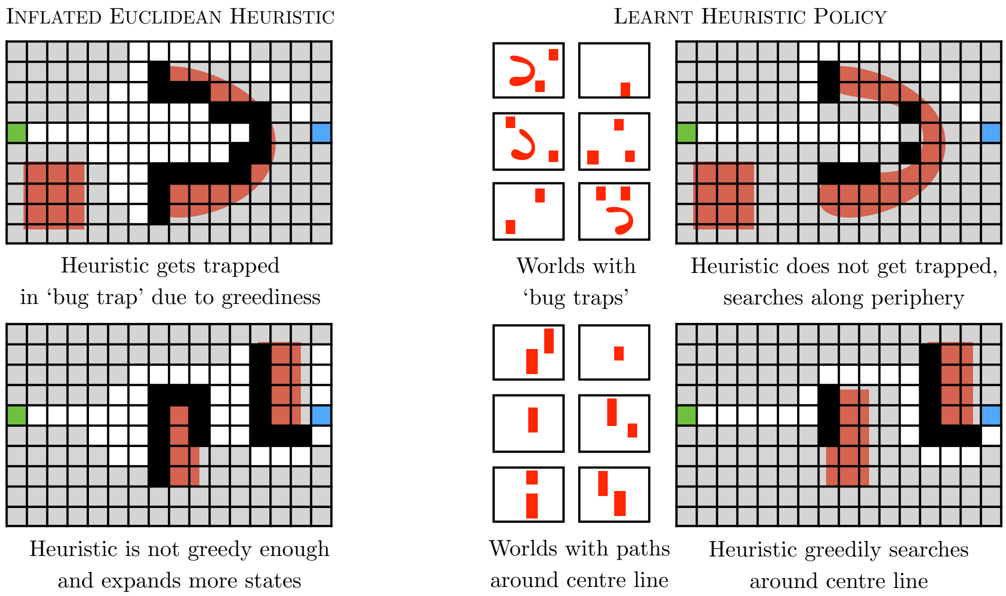


Figure 1: A learnt heuristic policy adapts to different obstacle configurations to minimize search effort. All schematics show the evolution of a search algorithm as the expansion of a search wavefront (expanded(white), invalid(black), unexpanded(grey)) from start (green) to goal (blue). A commonly used inflated euclidean heuristic cannot adapt to different environments, e.g it gets stuck in bugtraps. On the other hand, the learnt policy is able to infer the presence of a bug trap when trained on such a distribution and switch to greedy behaviour when trained on other distributions.

lead to a large search wavefront. Minimizing the estimation error does not necessarily minimize search effort.

Instead, we focus on the latter objective of designing heuristics that explicitly reduce search effort in the interest of real-time performance. Our key insight is that *heuristics should adapt during search* - as the search progresses, they should actively infer the structure of the valid configuration space, and focus the search on potentially good areas. Moreover, we want to learn this behaviour from data - changing the data distribution should change the heuristic automatically. Consider the example shown in Fig. 1. When a heuristic is trained on a world with 'bug traps', it learns to recognize when the search is trapped and circumvent it. On the other hand, when it is trained on a world with narrow gaps, it learns a greedy behaviour that drives the search to the goal.

To learn such behaviours, we propose a novel framework for training a data-driven heuristic policy that explicitly minimizes search effort. We formulate this as a sequential decision making problem where at a given iteration the heuristic policy uses only the information extracted from the wavefront to decide which node to expand, which in turn influences how the wavefront grows in the next iteration. We note that the training process for such partial information based policies is slow to converge and susceptible to local minima. We make a useful observation that if the heuristic has full information (which we call *clairvoyant planner*), it can use dynamic programming (Dijkstra) to efficiently compute the optimal decisions for any given wavefront. We leverage this property to train heuristics that imitate the clairvoyant planner during train time adopting the framework proposed by Choudhury et al. [9]. We make the following contributions:

1. We propose a novel framework to learn heuristic functions under the paradigm of sequential decision making under uncertainty.
2. We develop SAIL, an efficient algorithm for training heuristic functions by imitating clairvoyant oracles.
3. We demonstrate that we are able to learn heuristic policies with widely varying characteristics simply by training on different data distributions.

We note that a major limitation of this approach is that it ignores solution quality and we discuss several ways to alleviate this in Section 6.

Algorithm 1: $\texttt{Search}\langle v_s, v_g, \texttt{Succ}, \texttt{Eval}, \phi, \texttt{Select}\rangle$

```
𝒪 ← v_s, 𝒞 ← ∅, ℐ ← ∅;
while v_g ∉ 𝒪 do
    v ← Select(𝒪) ;                                          ▷ Select a vertex to expand
    (V_succ, E_inv) ← Expand(v, Succ, Eval, ϕ) ;             ▷ Invoke Succ(v) and Eval(e, ϕ)
    𝒪 ← 𝒪 ∪ V_succ, 𝒞 ← 𝒞 ∪ v, ℐ ← ℐ ∪ E_inv ;               ▷ Update all lists
return Path (v_s, v_g);
```

2 Preliminaries

2.1 Search Based Planning Framework

We consider the problem of search on a graph, $G = (V, E)$, where vertices V represent robot configurations and edges E represent potentially valid movements of the robot between these configurations. Given a pair of start and goal vertices, $(v_s, v_g) \in V$, the objective is to compute a path $\xi \subseteq E$ - a connected sequence of valid edges. The implicit graph G can be compactly represented by (v_s, v_g) and a successor function $\texttt{Succ}(v)$ which returns a list of outgoing edges and child vertices for a vertex $v \in V$. Hence a graph G is constructed during search by repeatedly *expanding* vertices using $\texttt{Succ}(v)$. Let $\phi \in \mathcal{M}$ be a representation of the world that is used to ascertain the validity of an edge. An edge $e \in E$ is checked for validity by invoking an evaluation function $\texttt{Eval}(e, \phi)$ which is an expensive operation and may require complex geometric intersection operations [10].

In this work, we focus on the *feasible path problem*. Alg. 1 defines a general search based planning algorithm $\texttt{Search}$ which takes as input the tuple $\langle v_s, v_g, \texttt{Succ}, \texttt{Eval}, \phi, \texttt{Select}\rangle$ and returns a valid path ξ. To ensure systematic search, the algorithm maintains the following lists - an open list $\mathcal{O} \subset V$ of candidate vertices to be expanded and a closed list $\mathcal{C} \subset V$ of vertices which have already been expanded. It also retains an additional invalid list $\mathcal{I} \subset V$ of edges found to be in collision. These 3 lists together represent the complete information available to the algorithm at any given point of time. At a given iteration, the algorithm uses this information to select a vertex $v \in \mathcal{O}$ to expand by invoking $\texttt{Select}(\mathcal{O})$. It then expands v by invoking $\texttt{Succ}(v)$ and checking validity of edges using $\texttt{Eval}(e, \phi)$ to get a set of valid successor vertices V_{succ} as well as invalid edges E_{inv}. The lists are then updated and the process repeated till the goal vertex v_g is uncovered.

2.2 Search as Sequential Decision Making under Uncertainty

We wish to learn an effective selection strategy $\texttt{Select}$ from data. We formalize this as a problem of sequentially making decisions (selecting vertices) under uncertainty (about the underlying world). We define a corresponding Markov Decision Process (MDP) [1] on the space of lists. At timestep t, let $s_t \in \mathcal{S}$ be the state of the search that is a concatenation of all lists, i.e $s_t = \{\mathcal{O}, \mathcal{C}, \mathcal{I}\}$. The action $a_t \in \mathcal{A}$ is the vertex $v \in \mathcal{O}$ that is selected by the search. On executing a_t, the new state s_{t+1} is determined by the underlying world ϕ. The world ϕ is a hidden variable, sampled from a prior $P(\phi)$ which in turn induces a state transition distribution $P(s_{t+1}|s_t, a_t)$. The one-step cost $c\,(s_t, a_t)$ is defined to be 1 for every (s_t, a_t) until the goal is added to the open list. Let $\pi(s_t)$ be a policy that maps state s_t to an action a_t. The policy represents the vertex selection strategy that we wish to learn. We term this policy as the *heuristic* guiding the search in a *best-first* fashion towards the goal. An episode continues till either v_g is selected or time horizon T is reached.

Given a prior distribution over worlds $P\,(\phi)$ and a distribution over start and goal vertices $P\,(v_s, v_g)$, we can evaluate the performance of a policy as

$$J\,(\pi) = \mathbb{E}_{\substack{\phi \sim P(\phi), \\ (v_s, v_g) \sim P(v_s, v_g)}} \left[\sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_\pi^t} \left[c\,(s_t, \pi(s_t)) \right] \right] \tag{1}$$

where $d_\pi^t = P\,(s_t|\pi, \phi, v_s, v_g)$ is the distribution over states induced by running π on the problem (ϕ, v_s, v_g) for t steps [11]. Our objective is to learn a policy

$$\pi^* = \underset{\pi \in \Pi}{\arg\min}\, J\,(\pi) \tag{2}$$

[1] Actually a POMDP which is an MDP over beliefs, referred here as an MDP over states for clarity.

3 Approach

3.1 Overview

An exact solution to the problem in (2) is intractable given the large state space ($|\mathcal{S}| = |V|$) and complex transition function $P(s_{t+1}|s_t, a_t)$. An alternative is to employ model-free Q-learning [12]. These methods try to minimize the cost-to-go $Q_t^\pi(s_t, a_t)$, i.e, the cumulative cost after executing a_t from s_t and subsequently executing policy π till the end of the episode. In general, such methods are not very sample efficient, slow to converge and additionally require training strategies such as experience replay and target networks[13].

However, we leverage a key insight for search based planning problems - if only we had full knowledge of the world ϕ, we could efficiently compute the optimal action from any state using dynamic programming. While we do not know the world at test time, we know it at train time - we can learn a policy to imitate this action. Hence we present the *Search as Imitation Learning (*SAIL*)* algorithm, a simple data-driven imitation learning approach for learning a heuristic best-first search policy.

3.2 Imitation Learning with Clairvoyant Oracles

Imitation of reference policies (or oracle policies) is a useful approach in scenarios where there exist good reference policies for the original problem, however these policies cannot be executed online (e.g due to computational complexity) hence requiring imitation via an offline training phase. Ross and Bagnell [11] use this idea to approach reinforcement learning problems for which there exists good yet expensive *model-based* oracle policies that cannot be executed at run-time but can be imitated by *model-free* policies. Hence they show a novel reduction of reinforcement learning to iterative supervised learning where the labels are the cost-to-go of the oracle [2] . Choudhury et al. [9] extend this idea to approach POMDP problems (specifically MDPs with a hidden variable) where exists good *clairvoyant oracles* - oracles that could solve the underlying MDP if only they could observe it fully. While such oracles cannot be executed at test time due to an information barrier - they can be imitated in a similar manner.

We adopt the framework of Choudhury et al. [9] as we too have an MDP whose transition function depends on a hidden world ϕ. We note that we can define an analogous *clairvoyant oracle planner* that employs a *backward* Dijkstra's algorithm, which given a world ϕ and a goal vertex v_g plans for the optimal path from every $v \in V$ using dynamic programming [3].

Definition 1 (Clairvoyant Oracle Planner)**.** Given full access to world ϕ and a goal v_g, the oracle planner encodes the cost-to-go from any vertex $v \in V$ as the function $Q^{\text{COR}}(v, \phi)$ which implicitly defines an oracle policy, $\pi_{\text{OR}}(s, \phi) = \underset{v \in \mathcal{O}}{\arg\min}\, Q^{\text{COR}}(v, \phi)$.

The clairvoyant oracle planner provides a look-up table $Q^{\text{COR}}(v, \phi)$ for the optimal cost-to-go from any vertex irrespective of the current state of the search. We define imitation of such an oracle as the following cost sensitive classification:

$$\hat{\pi}(s) = \underset{\pi \in \Pi}{\arg\min}\, \mathbb{E}_{\substack{\phi \sim P(\phi) \\ t \sim \mathcal{U}(1 \ldots T) \\ s \sim d_\pi^t}} \left[Q^{\text{COR}}(\pi(s), \phi) - \underset{v \in \mathcal{O}}{\arg\min}\, Q^{\text{COR}}(v, \phi) \right] \tag{3}$$

Intuitively, the term inside the expectation in Eq. 3 scores the learner's mis-classifications (incorrect vertex expansions) by how much additional future cost the oracle would incur if it chose the same action instead following its own policy. Given a world $\phi \sim P(\phi)$, if a policy π is executed upto a uniformly sampled timestep *t*, this scoring metric implicitly induces a ranking among all the states in the resulting open-list.

In order to learn the above policy, we use a reduction of cost-sensitive classification to $argmin$ *regression*. Our aim is to learn a parameterized function $Q_{\hat{\theta}}(s_t, a_t)$ that takes the current state and action as input and approximates $Q^{\text{COR}}(v, \phi)$, where a_t is $\texttt{Expand}(v, \texttt{Succ}, \texttt{Eval}, \phi)$. Using the

[2] A search state has several potentially good options. DAGGER [14], which uses a 0-1 loss, tries to solve a much harder learning problem of distinguishing amongst good options while AGGREVATE [11] focuses only on differentiating between good and bad options.

[3] For scaling to higher-dimensional planning, Random Geometric Graphs [15] can be used.

Algorithm 2: SAIL $(P(\phi), P(v_s, v_g), \beta_0, k)$

Initialize $\mathcal{D} \leftarrow \emptyset,\ \hat{\pi}_1$ to any policy in Π
for $i = 1, \ldots, N$ **do**
  Initialize sub-dataset $\mathcal{D}_i \leftarrow \emptyset$
  Collect *mk* datapoints as follows:
  for $j = 1, \ldots, m$ **do**
    Sample $\phi \sim P(\phi)$;
    Sample $(v_s, v_g) \sim P(v_s, v_g)$;
    Invoke clairvoyant oracle planner to compute $Q^{\text{COR}}(v, \phi)\, \forall v \in V$;
    Sample uniformly k timesteps $\{t_1, t_2, \ldots, t_k\}$ where each $t_i \in \{1, \ldots, T\}$;
    Rollout search with $\pi_{\text{mix}} = \beta_i \pi_{\text{OR}} + (1 - \beta_i)\hat{\pi}_i$; ▷ `Alg.1 with Select as` π_{mix}
    At each $t \in \{t_1, t_2, \ldots, t_k\}$ pick a random action a_t to get corresponding (v, s_t);
    Query oracle for $Q^{\text{COR}}(v, \phi)$; ▷ `Look-up optimal cost-to-go`
    $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \langle v, s_t, Q^{\text{COR}}(v, \phi) \rangle$; ▷ `Collect data for Eq.5`
    Continue roll-out with π_{mix} till end of episode.;
  Append to c.s classification data $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$;
  Train $\hat{\theta}_{i+1}$ on $\mathcal{D}$ to get $\hat{\pi}_{i+1}$;
return *Best* $\hat{\pi}$ *on validation*;

learnt parameters $\hat{\theta} \in \Theta$, the planner follows a greedy policy given by,

$$\hat{\pi}(s_t) = \arg\min_{a_t \in \mathcal{A}} Q_{\hat{\theta}}(s_t, a_t) \tag{4}$$

where $\hat{\theta}$ is learnt using the following procedure

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \mathbb{E}_{\substack{\phi \sim P(\phi) \\ t \sim \mathcal{U}(1 \ldots T) \\ s \sim d_\pi^t}} \left[\left(Q_\theta(s_t, a_t) - Q^{\text{COR}}(v, \phi) \right)^2 \right] \tag{5}$$

It is important to note that data for the learning the policy (Eq. 3 and Eq. 5) needs to be collected on the true distribution of states, d_π^t induced by executing policy π on world ϕ. A key distinction between our framework and that of Choudhury et al. [9] is that we directly get the cost-to-go value for all states by dynamic programming - we do not need to repeatedly invoke the oracle. We exploit this fact by extracting multiple labels from an episode even though the oracle is invoked only once.

3.3 SAIL Algorithm

Alg. 2, describes the SAIL framework which iteratively trains a sequence of policies $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$. For the optimization procedure described in Eq. 5, we collect a dataset $\mathcal{D}$ as follows - At every iteration *i*, the agent executed *m* different searches (Alg. 1). For every search, a different world ϕ and the pair (v_s, v_g) is sampled from a database. The agent then rolls-out a search with a mixture policy π_{mix} which blends the learner's current policy, $\hat{\pi}_i$ and the oracle's policy, π_{OR} using blending parameter β_i. During the search execution, at every timestep in a set of k uniformly sampled timesteps, we select a random action from the set of feasible actions and collect a datapoint $< v, s_t, Q^{\text{COR}}(v, \phi) >$. The policy π_{mix} is rolled out till the end of the episode and all the collected data is aggregated with dataset $\mathcal{D}$. The optimization in Eq. 5 can then be performed using either *online* or *mini-batch* learning on $\mathcal{D}$ to get the next policy $\hat{\pi}_{i+1}$.

$$\hat{\theta}_{i+1} = \arg\min_{\theta \in \Theta} \mathbb{E}_{(s_t, a_t, Q^{\text{COR}}) \sim \mathcal{D}} \left[\left(Q_\theta(s_t, a_t) - Q^{\text{COR}}(v, \phi) \right)^2 \right] \tag{6}$$

At the end of N iterations, the algorithm returns the best performing policy on a set of held-out validation environment or alternatively, a mixture of $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$. Note that while the oracle is invoked once per ϕ, we obtain k datapoints - this is critical for speeding up training.

We can obtain performance guarantees on the learnt policy directly applying analysis from [9]

Theorem 1. *The performance of the returned policy from Alg. 2 is, with probability at least* $1 - \delta$

$$J(\hat{\pi}) \leq J(\tilde{\pi}_{\text{OR}}) + 2\sqrt{|\mathcal{A}|}T \sqrt{\varepsilon_{\text{class}} + \varepsilon_{\text{reg}} + O\left(\sqrt{\log\left(({}^{1}/_{\delta})/Nm\right)}\right)} + O\left(\frac{Q_{\max}^{\text{COR}} T \log T}{\alpha N}\right)$$

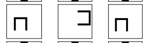
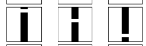
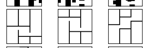
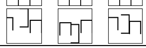
Dataset	Sample Worlds	SAIL	SL	CEM	QL	h_{EUC}	h_{MAN}	A*	MHA*
Alternating Gaps		**0.039**	0.432	0.042	1.000	1.000	1.000	1.000	1.000
Single Bugtrap		0.158	0.214	**0.057**	1.000	0.184	0.192	1.000	0.286
Shifting Gaps		**0.104**	0.464	1.000	1.000	0.506	0.589	1.000	0.804
Forest		**0.036**	0.043	0.048	0.121	0.041	0.043	1.000	0.075
Bugtrap+Forest		**0.147**	0.384	0.182	1.000	0.410	0.337	1.000	0.467
Gaps+Forest		**0.221**	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Mazes		**0.103**	0.238	0.479	0.399	0.185	0.171	1.000	0.279
Multiple Bugtraps		**0.479**	0.480	1.000	0.835	0.648	0.617	1.000	0.876

Figure 2: Normalized cost of baselines on different environments (best in bold). The cost corresponds to average expansions on a test set of planning problems normalized between [200, 5000] (max possible: 40000). Planning parameters are - map size: 200×200,$T_{train} = 1100$, $T_{test} = 20000$. Data sizes are: train(200), test(100), validation(70). SAIL parameters are - $k : 50, \beta_0 = 0.7$. SAIL, CEM and QL are run for $N : 15$ iterations. SL uses $m : 600$.

We note that even though the time complexity of `Select` is $O(|\mathcal{O}_t|)$ at timestep t, SAIL can have better overall complexity if it can achieve a squared reduction in number of expansions compared to uninformed search (refer to supplementary for discussion [4]).

4 Experiments

4.1 Implementation Details

We evaluate SAIL on a variety of 2D navigation tasks where the robot has to plan from bottom-left to top-right on an 8-connected grid. The grid is embedded on a binary map of obstacles. The function Q_θ is represented by a feed-forward neural network with two fully connected hidden layers containing [100, 50] units and ReLu activation. The model takes as input a 17 dimensional feature vector $f \in \mathcal{F}$ for the pair (v, s) which contains values like closest invalid state in $\mathcal{I}$, distance to start and goal, depth in the tree etc. Refer to supplementary for details [4].

4.2 Baseline Approaches For Heuristic Search

Motion Planning Baselines: We compare against greedy best-first search with 2 commonly used heuristics - the euclidean distance (h_{EUC}) and the manhattan distance (h_{MAN}). We also use A* algorithm as a baseline with h_{EUC} heuristic. Additionally, we compare against the MHA* algorithm [16] which has been proven to be an effective way of combining multiple, often un-related, heuristics providing bounds on solution quality [17]. We use a simplified version which expands three different heuristics in a round-robin fashion - $[h_{\text{EUC}}, h_{\text{MAN}}, d_{OBS}]$, where d_{OBS} is the euclidean distance to closest, *known* obstacle cell in $\mathcal{I}$.

Machine Learning Baselines: We consider two learning baselines (a) Supervised Learning (SL) with data from roll-outs with π_{OR} and (b) Reinforcement Learning using evolutionary strategies (CEM) and Q-Learning (QL) with function approximation. Refer to supplementary for details [4] .

4.3 Analysis of Results

Fig. 2 shows the normalized evaluation cost of all algorithms on various datasets. Snapshots of planning with different heuristics are shown in Fig. 3 and Fig. 4(a). Convergence of different learning algorithms are shown in Fig. 4(b). We present a set of observations.

O 1. SAIL *has a consistently competitive performance across all datasets.*

Fig. 2 shows that SAIL learns a better search policy than any other baseline across all but one environments. It maintains performance from homogenous to heterogenous environments.

O 2. SAIL *has faster convergence than all learning baselines.*

Fig. 4(b) shows that on the 'Forest' dataset, SAIL converges by 6^{th} iteration, while CEM takes 12 and QL does not converge. SAIL also converges quickly (by the 8^{th} iteration) across datasets.

[4] Source code for planning pipeline, an OpenAI Gym [18] environment and datasets along with supplementary material can be accessed via our project page at this link `https://goo.gl/YXkQAC`

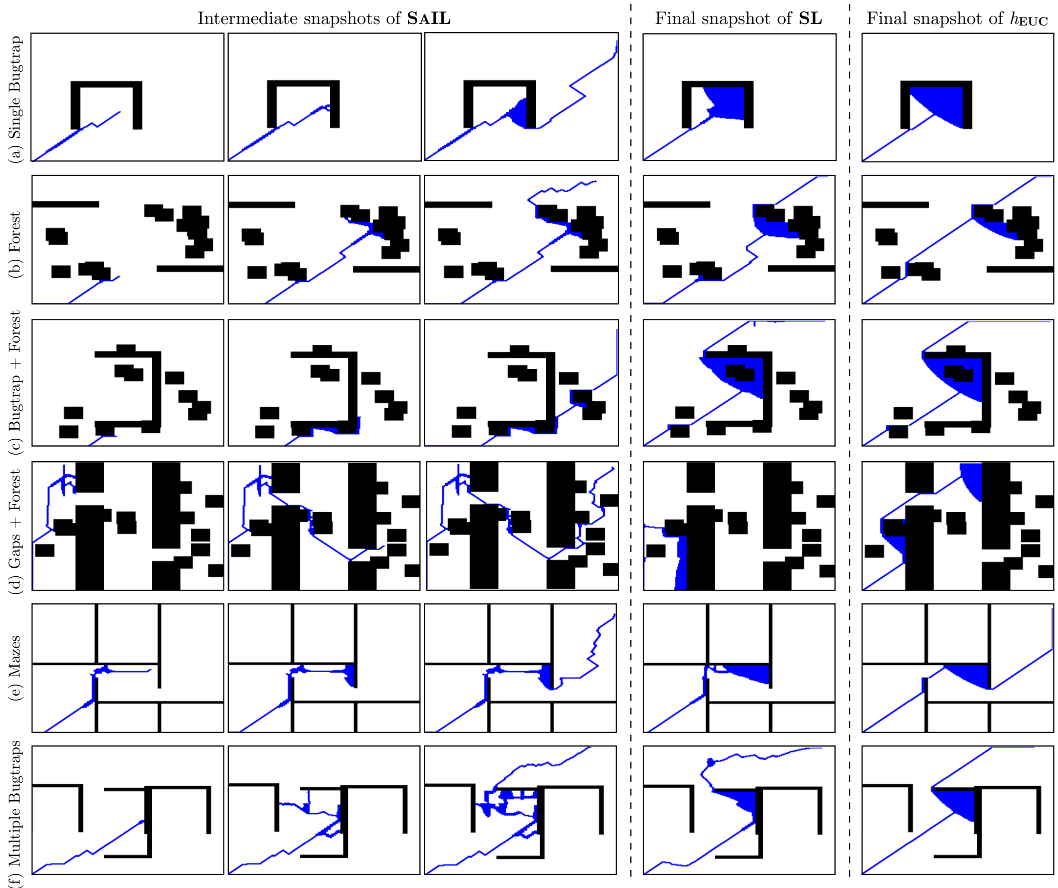


Figure 3: Evolution of search frontier (expanded(blue), invalid(black), unexpanded(white)) of SAIL compared with final snapshot of supervised learning (SL) and h_{EUC} across all environments. SAIL expands far less states.

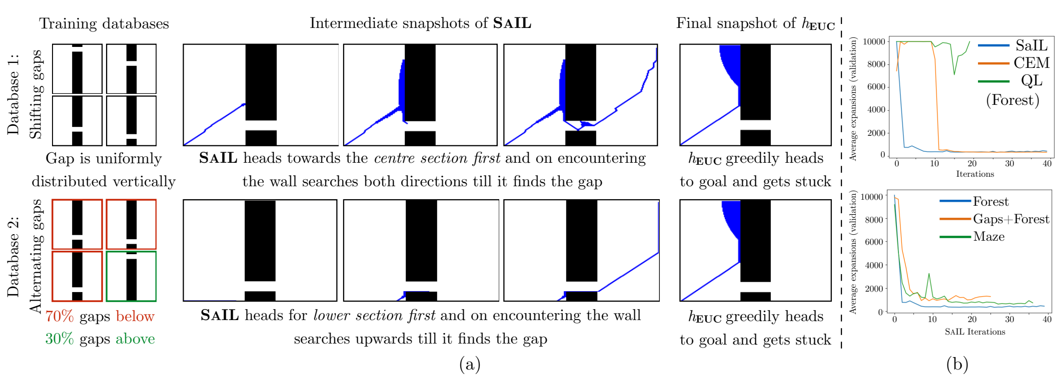


Figure 4: (a) SAIL learns to adapt to different environment distributions by directing search to areas where it expects to find gaps. Note SAIL does not have information about the entire environment, only the explored part. (b) On the 'Forest' dataset, SAIL converges faster that CEM and QL to a good policy. SAIL also converges consistently to a good policy across environments 'Gaps', 'Gaps+Forest', 'Maze.'

O 3. SAIL *adapts the behavior of the search in response to a change in world distribution* $P(\phi)$.
Fig. 4(a) shows an intuitive example of how simply changing the distribution of where gaps occur along a wall affects the way SAIL chooses to progress the search.

O 4. SAIL *is able to detect and escape local minima.*

A classic case in motion planning is the bugtrap (Fig. 1) which traps a greedy search in a local minimum. Fig. 3(a) and Fig. 3(f) shows that when trained on such distributions, SAIL is able to detect these artifacts and smartly escape them by exploring in different directions.

O 5. SAIL *is able to exploit the relative configuration of obstacles and environment structure.* In a maze world with rectilinear hallways (Fig. 3(e)), SAIL learns to quickly find a wall and then concentrate the search along the axes. In Fig. 3(d), SAIL focuses only on regions where there is a high probability of a gap and skids along obstacles otherwise.

5 Related Work

Learning heuristics falls under machine learning for general purpose planning [19]. Yoon et al. [20] [21] propose using regression to learn residuals over FF-Heuristic [22]. Xu et al. [23] [24, 25] improve upon this in a beam-search framework. Arfaee et al. [26] iteratively improve heuristics. ús Virseda et al. [27] learn combination of heuristic to estimate cost-to-go. Kendall rank coefficient is used to learn open list ranking [28, 29]. Thayer et al. [30] learn heuristics online during search. Paden et al. [8] learn admissible heuristics as S.O.S problems. However, these methods do not address minimization of search effort and also ignore the non i.i.d nature of the problem.

Relevant work in imitation learning examines the non i.i.d supervised learning problem of imitating oracles under one's own state distribution. Ross et al. [14], Ross and Bagnell [11] use dataset aggregation to reduce such problems to no-regret iterative supervised learning. Choudhury et al. [9, 31] apply such methods to learn information gathering policies. Recent deep reinforcement learning approaches also employ supervised learning by imitating oracles as they offer better sample efficiency and safety than model free policy search [32, 33, 34] or Q learning [13, 35]. Zhang et al. [36] extend guided policy search [37] for imitating MPC. Kahn et al. [38] further adapt the MPC expert to generate training sample for states likely to be visited. Tamar et al. [39] consider a cost-shaping approach for short horizon MPC by offline imitation of long horizon MPC which is closest to our work. Tamar et al. [40] develop a neural network architecture with an explicit planning component embedded in it. Gupta et al. [41] develop a holistic mapping and planner framework trained using feedback from optimal plans on a graph.

6 Discussion and Future Work

We now discuss some insights and directions for future work.

Q 1. *When do we expect this framework of imitating clairvoyant oracles to work?*

The analysis adopted from [9] states that the performance of the learnt policy using SAIL is near-optimal with respect to a *hallucinating oracle* - an oracle that hallucinates different worlds conditioned on the current open list and expands the best node. The hallucinating oracle is similar in nature to a QMDP algorithm [42], an effective approximate solution to POMDPs, which takes the best action on the current posterior. However, while QMDP is model based (requires an explicit posterior), SAIL is model free. QMDP has been shown to be very successful where explicit information gathering behaviour is not required [43, 44] - the belief collapses irrespective of the action. This is very apt in the problem we address - as the set of actions are constrained to candidate nodes in the open list, no single action is very informative. It suffices to expand the best node under the current belief and continue to update the belief as the open list evolves. We note that this is not true for all learning in planning paradigms. For example, when learning to collision check [45], a policy that actively reduces uncertainty about the world is effective.

Q 2. *How can we incorporate solution cost in addition to search effort in this framework?*

While our framework ignore the cost of a solution, we note that finding feasible solutions quickly is the core motivation of a number of high dimensional planning problems which have historically resorted to sampling based approaches [46]. Hence, one can apply our framework to such problems to produce potentially faster solutions. We also note that when planning on locally connected lattices for geometric planning problems, minimizing the number of expansions generally leads to near-optimal solutions (unit cost for each valid edge). However, if we really cared about near optimal solutions, the framework of Multi-heuristic A* (MHA*) [16] can be easily adopted. In such a framework, any heuristic function [47] can be used in tandem with an anchored search which uses an inflated admissible heuristic. Hence we can simply replace our `Search` function with MHA*. The bi-objective criteria of solution cost and search effort is best reasoned about in the paradigm of *anytime planning*. In this paradigm, an algorithm traces out the *pareto-frontier* [48] - finds a feasible solution quickly and iteratively improves it. In this paradigm, SAIL trains a heuristic that displays a behaviour we would expect in the first iteration. A direction of future work would be to learn *anytime heuristics* that minimize search effort initially to and solution cost eventually.

Acknowledgments

We would like to acknowledge the support from ONR grant N000141310821 and NASA contract NNX17CS56C.

References

[1] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[2] J. Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

[3] C. M. Dellin, K. Strabala, G. C. Haynes, D. Stager, and S. S. Srinivasa. Guided manipulation planning at the darpa robotics challenge trials. In *Experimental Robotics*, 2016.

[4] H. Cover, S. Choudhury, S. Scherer, and S. Singh. Sparse tangential network (spartan): Motion planning for micro aerial vehicles. In *ICRA*. IEEE, 2013.

[5] I. Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 1970.

[6] M. Likhachev and D. Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 2009.

[7] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Practical search techniques in path planning for autonomous driving. *AAAI*, 2008.

[8] B. Paden, V. Varricchio, and E. Frazzoli. Verification and synthesis of admissible heuristics for kinodynamic motion planning. *IEEE Robotics and Automation Letters*, 2(2):648–655, 2017.

[9] S. Choudhury, A. Kapoor, G. Ranade, S. Scherer, and D. Dey. Adaptive information gathering via imitation learning. In *RSS*, 2017.

[10] C. M. Dellin and S. 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.

[11] S. Ross and J. A. Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv:1406.5979*, 2014.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[14] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[15] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[16] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev. Multi-heuristic a. *The International Journal of Robotics Research*, 2016.

[17] M. Phillips, V. Narayanan, S. Aine, and M. Likhachev. Efficient search with an ensemble of heuristics. In *IJCAI*, 2015.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[19] S. Jiménez, T. De La Rosa, S. Fernández, F. Fernández, and D. Borrajo. A review of machine learning for automated planning. *The Knowledge Engineering Review*, 2012.

[20] S. W. Yoon, A. Fern, and R. Givan. Learning heuristic functions from relaxed plans. In *ICAPS*, 2006.

[21] S. Yoon, A. Fern, and R. Givan. Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, 2008.

[22] J. Hoffmann and B. Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 2001.

[23] Y. Xu, A. Fern, and S. W. Yoon. Discriminative learning of beam-search heuristics for planning. In *IJCAI*, 2007.

[24] Y. Xu, A. Fern, and S. W. Yoon. Iterative learning of weighted rule sets for greedy search. In *ICAPS*, 2010.

[25] Y. Xu, A. Fern, and S. Yoon. Learning linear ranking functions for beam search with application to planning. *Journal of Machine Learning Research*, 2009.

[26] S. J. Arfaee, S. Zilles, and R. C. Holte. Learning heuristic functions for large state spaces. *Artificial Intelligence*, 2011.

[27] J. ús Virseda, D. Borrajo, and V. Alcázar. Learning heuristic functions for cost-based planning. *Planning and Learning*, 2013.

[28] C. M. Wilt and W. Ruml. Building a heuristic for greedy search. In *Eighth Annual Symposium on Combinatorial Search*, 2015.

[29] C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez. Learning to rank for synthesizing planning heuristics.

[30] J. T. Thayer, A. J. Dionne, and W. Ruml. Learning inadmissible heuristics during search. In *ICAPS*, 2011.

[31] S. Choudhury, A. Kapoor, G. Ranade, and D. Dey. Learning to gather information via imitation. In *ICRA*, 2017.

[32] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *ICML*, 2015.

[33] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[34] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[35] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *ICML*, 2016.

[36] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *ICRA*, 2016.

[37] S. Levine and V. Koltun. Guided policy search. In *ICML*, 2013.

[38] G. Kahn, T. Zhang, S. Levine, and P. Abbeel. Plato: Policy learning using adaptive trajectory optimization. In *ICRA*, 2017.

[39] A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel. Learning from the hindsight plan–episodic mpc improvement. *arXiv preprint arXiv:1609.09001*, 2016.

[40] A. Tamar, S. Levine, and P. Abbeel. Value iteration networks. *NIPS*, 2016.

[41] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017.

[42] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, 1995.

[43] M. Koval, N. Pollard, and S. Srinivasa. Pre- and post-contact policy decomposition for planar contact manipulation under uncertainty. In *RSS*, 2014.

[44] S. Javdani, S. S. Srinivasa, and J. A. Bagnell. Shared autonomy via hindsight optimization. In *RSS*, 2015.

[45] S. Choudhury, S. Javdani, S. Srinivasa, and S. Scherer. Near-optimal edge evaluation in explicit generalized binomial graphs. In *NIPS*, 2017.

[46] J. J. Kuffner and S. M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *ICRA*, 2000.

[47] V. Narayanan, S. Aine, and M. Likhachev. Improved multi-heuristic a* for searching with uncalibrated heuristics. In *Eighth Annual Symposium on Combinatorial Search*, 2015.

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