Article

The International Journal of Robotics Research
2018, Vol. 37(13-14) 1632–1672

DOI: 10.1177/0278364918781001
journals.sagepub.com/home/ijr

Data-driven planning via imitation learning

Sanjiban Choudhury[1], Mohak Bhardwaj[1], Sankalp Arora[1], Ashish Kapoor[2], Gireeja Ranade[2], Sebastian Scherer[1] and Debadeepta Dey[2]

Abstract

Robot planning is the process of selecting a sequence of actions that optimize for a task=specific objective. For instance, the objective for a navigation task would be to find collision-free paths, whereas the objective for an exploration task would be to map unknown areas. The optimal solutions to such tasks are heavily influenced by the implicit structure in the environment, i.e. the configuration of objects in the world. State-of-the-art planning approaches, however, do not exploit this structure, thereby expending valuable effort searching the action space instead of focusing on potentially good actions. In this paper, we address the problem of enabling planners to adapt their search strategies by inferring such good actions in an efficient manner using only the information uncovered by the search up until that time. We formulate this as a problem of sequential decision making under uncertainty where at a given iteration a planning policy must map the state of the search to a planning action. Unfortunately, the training process for such partial-information-based policies is slow to converge and susceptible to poor local minima. Our key insight is that if we could fully observe the underlying world map, we would easily be able to disambiguate between good and bad actions. We hence present a novel data-driven imitation learning framework to efficiently train planning policies by imitating a clairvoyant oracle*: an oracle that at train time has full knowledge about the world map and can compute optimal decisions. We leverage the fact that for planning problems, such oracles can be efficiently computed and derive performance guarantees for the learnt policy. We examine two important domains that rely on partial-information-based policies: informative path planning and search-based motion planning. We validate the approach on a spectrum of environments for both problem domains, including experiments on a real UAV, and show that the learnt policy consistently outperforms state-of-the-art algorithms. Our framework is able to train policies that achieve up to* 39% *more reward than state-of-the art information-gathering heuristics and a* 70× *speedup as compared with A* on search-based planning problems. Our approach paves the way forward for applying data-driven techniques to other such problem domains under the umbrella of robot planning.*

Keywords

Imitation learning, POMDPs, sequential decision making, QMDPs, heuristic search

1. Introduction

Motion planning, the task of computing a sequence of collision-free motions for a robotic system from a start to a goal configuration, has a rich and varied history (LaValle, 2006). Up until now, the bulk of the prominent research has focused on the development of tractable planning algorithms with provable *worst-case performance guarantees* such as computational complexity (Canny, 1988), probabilistic completeness (LaValle and Kuffner, 2001), or asymptotic optimality (Karaman and Frazzoli, 2011). In contrast, analysis of the *expected performance* of these algorithms on real-world planning problems a robot encounters has received considerably less attention, primarily due to the lack of standardized datasets or robotic platforms.

Informative path planning (IPP), the task of computing an optimal sequence of sensing locations to visit so as to maximize information gain, has also had an extensive amount of prior work on algorithms with provable worst-case performance guarantees such as computational complexities (Singh et al., 2007) and the probabilistic completeness (Hollinger and Sukhatme, 2013) of information theoretic planning. Although these algorithms use heuristics to

[1]Carnegie Mellon University, Pittsburgh, PA, USA
[2]Microsoft Research, Redmond, WA, USA

Corresponding author:
Sanjiban Choudhury, Carnegie Mellon University, Pittsburgh, PA 15217, USA.
Email: sanjiban@cmu.edu

approximate information gain using variants of Shannon's entropy, their expected performance on real-world planning problems is heavily influenced by the geometric distribution of objects encountered in the world.

A unifying theme for both these problem domains is that as robots break out of contrived laboratory settings and operate in the real world, the scenarios encountered by them vary widely and have a significant impact on performance. Hence, a key requirement for autonomous systems is a *robust planning module* that maintains *consistent performance* across the diverse range of scenarios it is likely to encounter. To do so, planning modules must possess the ability to leverage information about the implicit structure of the world in which the robot operates and adapt the planning strategy accordingly. Moreover, this must occur in a pure *data-driven fashion* without the need for human intervention. Fortunately, recent advances in affordable sensors and actuators have enabled mass deployment of robots that navigate, interact, and collect real data. This motivates us to examine the following question:

> How can we design planning algorithms that, subject to on-board computation and sensing constraints, maximize their expected performance on the actual distribution of problems that a robot encounters?

1.1. Motivation

We look at two domains: IPP and search-based planning. We briefly delve into these problems and make the case for data-driven approaches in both.

1.1.1. IPP. We consider the following information-gathering problem: given a hidden world map, sampled from a prior distribution, the goal is to successively visit sensing locations such that the amount of relevant information uncovered is maximized while not exceeding a specified fuel budget. This problem fundamentally recurs in mobile robot applications such as autonomous mapping of environments using ground and aerial robots (Charrow et al., 2015; Heng et al., 2015), monitoring of water bodies (Hollinger and Sukhatme, 2013) and inspecting models for 3D reconstruction (Hollinger et al., 2017; Isler et al., 2016).

The nature of "interesting" objects in an environment and their spatial distribution influence the optimal trajectory a robot might take to explore the environment. As a result, it is important that a robot learns about the type of environment it is exploring as it acquires more information and adapts its exploration trajectories accordingly.

To illustrate our point, we sketch out two extreme examples of environments for a particular mapping problem, shown in Figure 1(a). Consider a robot equipped with a sensor (RGBD camera) that needs to generate a map of an unknown environment. It is given a prior distribution about the geometry of the world, but has no other information. This geometry could include very diverse settings. First it can include a world where there is only one ladder, but the form of the ladder must be explored, which is a very dense setting. Second, it could include a sparse setting with spatially distributed objects, such as a construction site.

The important task for the robot is to now try to infer which type of environment it is in based on the history of measurements, and thus plan an efficient trajectory. At every timestep, the robot visits a sensing location and receives a sensor measurement (e.g. depth image) that has some amount of information utility (e.g. surface coverage of objects with point cloud). As opposed to naive lawnmower-coverage patterns, it will be more efficient if the robot could use a policy that maps the history of locations visited and measurements received to decide which location to visit next such that it maximizes the amount of information gathered in the finite amount of battery time available.

The ability of such a learnt policy to gather information efficiently depends on the prior distribution of worlds in which the robot has been shown how to navigate optimally. Figure 1(a) (left) shows an efficient learnt policy for inspecting a ladder, which executes a helical motion around parts of the ladder already observed to efficiently uncover new parts without searching naively. This is efficient because given the prior distribution the robot learns that information is likely to be geometrically concentrated in a particular volume given its initial observations of parts of the ladder. Similarly, Figure 1(a) (right) shows an effective policy for exploring construction sites by executing large sweeping motions. Here again the robot learns from prior experience that wide, sweeping motions are efficient since it has learnt that information is likely to be dispersed in such scenarios. We wish to arrive at an efficient procedure for training such a policy.

1.1.2. Search-based planning. Search-based motion planning offers a comprehensive framework for reasoning about a vast number of motion planning algorithms (LaValle, 2006). In this framework, an algorithm grows a *search tree* of feasible robot motions from a start configuration towards a goal (Pearl, 1984). 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 until it finds the goal as illustrated in Figure 1(b).

This paper addresses a class of robotic motion planning problems where edge evaluation dominates the search effort, such as for robots with complex geometries such as robot arms (Dellin et al., 2016) or for robots with limited onboard computation such as unmanned aerial vehicles (UAVs) (Cover et al., 2013). To ensure real-time performance, algorithms must prioritize minimizing the search

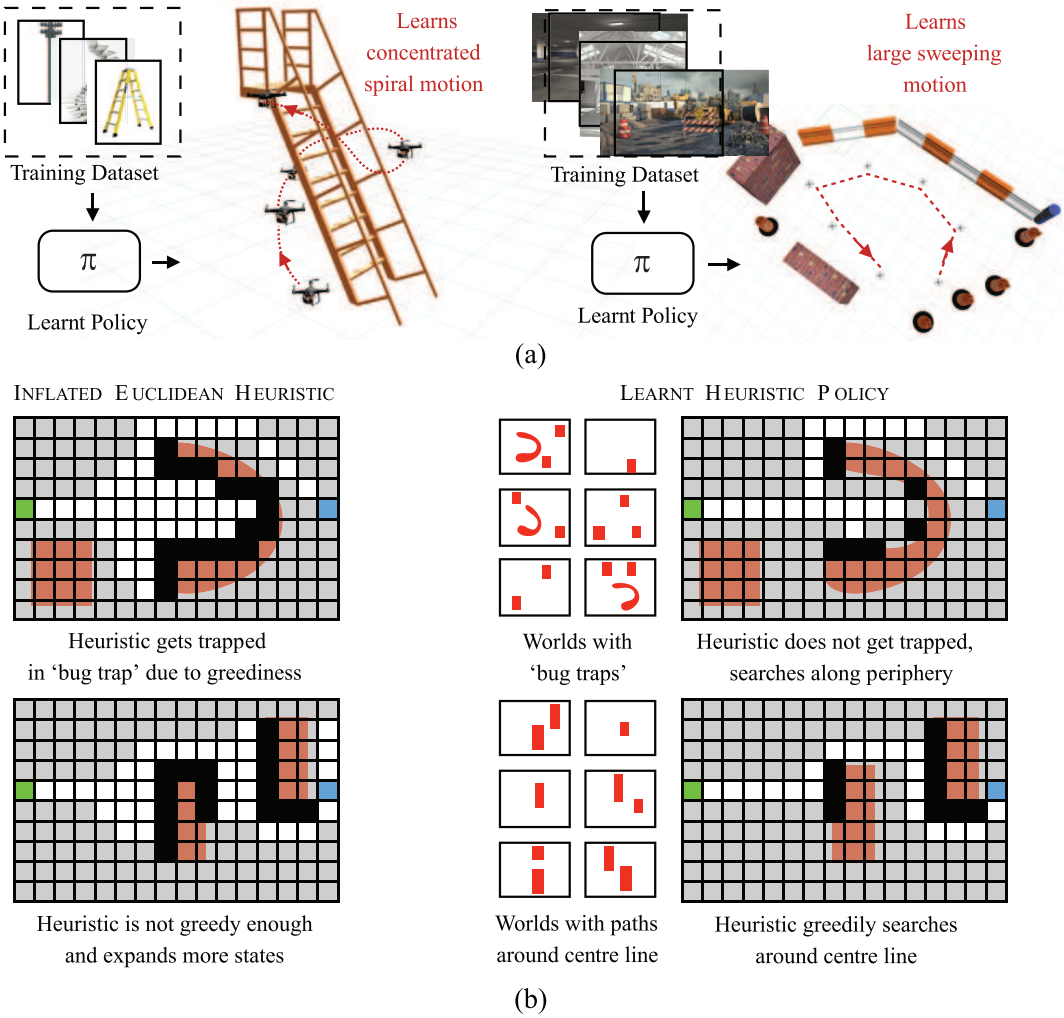


Fig. 1. Sequential decision making in IPP and search-based planning. The implicit structure of the environment affects the performance of policies in both tasks. (a) The effectiveness of a policy to gather information depends on the distribution of worlds. (Left) When the distribution corresponds to a scene containing ladders, the learnt policy executes a helical motion around parts of the ladder already observed as it is unlikely that there is information elsewhere. (Right) When the distribution corresponds to a scene from a construction site, the learnt policy executes a large sweeping motion as information is likely to be dispersed. (b) 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 behavior when trained on other distributions.

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 Figure 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 focused on the former objective of guaranteeing that the search returns a near-optimal solution (Pearl, 1984). These approaches define a heuristic function as a *distance metric* that estimates the cost-to-go value of a node (Pohl, 1970). However, estimation of this distance metric is difficult as it is 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. Although state-of-the-art methods propose different relaxation-based (Dolgov et al., 2008; Likhachev and Ferguson, 2009) and learning-based approaches (Paden et al., 2017) to estimate the distance metric they run into a much more fundamental

limitation: *a small estimation error can 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 behavior from data: changing the data distribution should change the heuristic automatically. Consider the example shown in Figure 1(b). 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 behavior that drives the search to the goal.

1.2. Key idea

It is natural to think of both these problems as a partially observable Markov decision process (POMDP). However the POMDP is defined on a belief over possible world maps rendering even the most efficient of online POMDP solvers impractical.

Our key insight is that if the policies could fully observe and process the world map during decision making, they could quite easily disambiguate good actions from bad ones. This motivates us to frame the problem of learning a planning policy as a novel data-driven imitation (Ross and Bagnell, 2014) of a *clairvoyant oracle*. During the training process, the oracle has full knowledge about the world map (hence, clairvoyant) and selects actions that maximize cumulative rewards. The policy is then trained to imitate these actions as best as it can using partial knowledge from the current history of actions and observations. As a result of our novel formulation, we are able to sidestep a number of challenging issues in POMDPs such as explicitly computing posterior distribution over worlds and planning in belief space.

We empirically show that training such policies using imitation learning (IL) of clairvoyant oracles leads to much faster convergence and robustness to poor local minima than training policies via model-free policy improvement. We leverage the fact that such oracles can be efficiently computed for our domains once the source of uncertainty is removed. We show in our analysis that imitation of such clairvoyant oracles during training is equivalent to being competitive with a *hallucinating oracle* at test time, i.e. an oracle that implicitly maintains a posterior over world maps and selects the best action at every timestep. This offers some valuable insight behind the success of this approach as well as instances where such an approach would lead to a near-optimal policy.

1.3. Contributions.

Our contributions are as follows.

1. We motivate the need to learn a planning policy that adapts to the environment in which the robot operates. We examine two domains: IPP and search-based planning. We examine both problems through the lens of sequential decision making under uncertainty (Section 2).
2. We present a novel mapping of both these problems to a common POMDP framework (Section 3).
3. We propose a novel framework for training such POMDP policies via IL of a clairvoyant oracle. We analyze the implications of imitating such an oracle (Section 4).
4. We present training procedures that deal with the non-independent and identically distributed (non-i.i.d.) distribution of states induced by the policy itself along with performance guarantees. We present concrete instances of the algorithm for both problem domains. We also show that for a certain class of IPP problems, policies trained in this fashion possess near-optimality properties (Section 5).
5. We extensively evaluate the approach on both problem domains. In each domain, we evaluate on a spectrum of environments and show that policies outperform state-of-the-art approaches by exhibiting adaptive behaviors. We also demonstrate the impact of this framework on real-world problems by presenting flight test results from a UAV (Sections 6 and 7).

This paper is an unification of previous works on adaptive information gathering (Choudhury et al., 2017b,c) and learning heuristic search (Bhardwaj et al., 2017). We present a unified framework for reasoning about both problems. We compare and contrast training procedures due to both domains. We present new results in learning heuristics on 4D planning problems and present flight test results from a UAV. We present new results on comparing the IL with policy search and comparing sample efficiency of AGGREVATE and FORWARDTRAINING. We present more details on implementation and analysis of results. We provide comprehensive discussions on shortcomings of this approach and directions for future work in Section 8.

2. Background

In this section, we present some background on the two domains that we examine: IPP in Section 2.4 and search-based planning in Section 2.2. The formal problem that we wish to solve in each of these domains is defined. We also present some background on POMDPs in Section 2.3 and IL in Section 2.4 that forms the basis of our approach.

2.1. IPP

We now present a framework for IPP where the objective is to visit maximally informative sensing locations subjected to time and travel constraints. We use this framework to pose the problem of computing an information-gathering policy for a given distribution over worlds and briefly discuss prior work on this topic.

2.1.1. Framework. We now introduce a framework and set of notation to express the IPP problems of interest. The specific implementation details of the problem are described in detail in Section 6.1.

We have a robot that is constrained to move on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes corresponding to all sensing locations. The start node is v_s. Let $\xi = (v_1, v_2, \ldots, v_p)$ be a sequence of connected nodes (a path) such that $v_1 = v_s$. Let Ξ be the set of all such paths.

Let $\phi \in \mathcal{M}$ be the world map in which the robot operates. The world map is usually represented in practice as a binary grid map where grid cells are either occupied or free. We assume that the world map is fixed during an episode.

Let $y \in \mathcal{Y}$ be a measurement received by the robot. Let $\mathcal{H} : \mathcal{V} \times \mathcal{M} \to \mathcal{Y}$ be a measurement function. When the robot is at node v in a world map ϕ, the measurement y received by the robot is $y = \mathcal{H}(v, \phi)$. The measurement function is defined by a sensor model, e.g. a range-limited sensor. A measurement is obtained by projecting the sensor model onto the sensing node v and ray-casting to determine the surfaces of the underlying world ϕ that intersect with the sensor rays.

The objective of the robot is to move on the graph and maximize utility. Let $\mathcal{F} : 2^{\mathcal{V}} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$ be a utility function. For a set of vertices ξ and a world map ϕ, $\mathcal{F}(\xi, \phi)$ assigns a utility to visiting the vertices in the world. The utility of a measurement from a node is usually the amount of surface of the world covered by it. In such an instance, the function does not depend on the sequence of vertices in the path, i.e. is a set function. For simplicity, we assume that the measurement and utility function is deterministic. However, this assumption can easily be relaxed in our approach and is discussed in Section 8.4.

As the robot moves on the graph, the travel cost is captured by the cost function $\mathcal{T} : \Xi \times \mathcal{M} \to \mathbb{R}_{\geq 0}$. For a path ξ and a world map ϕ, $\mathcal{T}(\xi, \phi)$ assigns a travel cost for executing the path on the world. In a practical setting, the total number of timesteps is bounded by T and the travel cost is bounded by B. Figure 2 shows an illustration of the framework.

We are now ready to define the IPP problems. There are two axes of variations:

1. constraint on the motion of the robot;
2. observability of the world map.

The first axis arises from whether the robot is subject to any travel constraints. For problems such as sensor placement, the agent is free to select any sequence of nodes and the travel cost between nodes is zero. For such situations, the graph is also fully connected to permit any sequence. For problems involving physical movements, the agent is constrained by a budget on the travel cost. In addition, the graph may also not be fully connected.

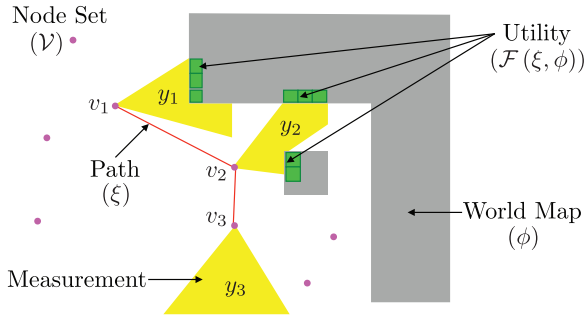


Fig. 2. The IPP problem. Given a world map ϕ, the robot plans a path ξ that visits a node $v_i \in \mathcal{V}$ and receives measurement y_i, such that utility (information gathered) $\mathcal{F}(\xi, \phi)$ is maximized. Here the utility is the cardinality of all the cells uncovered (green), which is a union of the cells uncovered at each location (and hence a set cover function).

The second axis arises from different task specifications which result in the world map being observable or being hidden. We categorize the problems on this axis to aid future discussions on imitating clairvoyant oracles in Section 5.

2.1.2. Problems with known world maps. For the first two variants, the world map ϕ is known and can be evaluated while computing a path ξ. These problems arise in applications involving inspection of known structures (such as the hull of a ship): the robot has a model of the object but still needs to obtain measurements of the surface for analysis of structural damages.

Problem 1 (KNOWN-UNC: known world map; unconstrained travel cost)**.** *Given a world map ϕ, a fully connected graph $\mathcal{G}$ and a time horizon T, find a path ξ that maximizes utility*

$$\begin{aligned} \arg\max_{\xi \in \Xi} \quad & \mathcal{F}(\xi, \phi) \\ s.t. \quad & |\xi| \leq T + 1 \end{aligned} \tag{1}$$

In the case where the utility function is a set function, Problem 1 is a set function maximization problem which in general can be NP-hard (Krause and Golovin, 2012)). Such problems occur commonly in the sensor-placement problem Krause et al. (2008). However, in many instances the utility function can be shown to possess the powerful property of *monotone submodularity*. This property implies the following.

1. *Monotonic improvement*: The value of the utility can only increase on adding nodes, i.e.

$$\mathcal{F}(\mathcal{V}_1 \cup \mathcal{V}_2, \phi) \geq \mathcal{F}(\mathcal{V}_1, \phi)$$

for all $\mathcal{V}_1, \mathcal{V}_2 \subseteq \mathcal{V}$

2. *Diminishing returns*: The gain in adding a set of nodes diminishes with increasing nodes

$$\mathcal{F}(\mathcal{V}_1 \cup \mathcal{V}_3, \phi) - \mathcal{F}(\mathcal{V}_3, \phi) \leq \mathcal{F}(\mathcal{V}_1 \cup \mathcal{V}_2, \phi) - \mathcal{F}(\mathcal{V}_2, \phi)$$

for all $\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3 \subseteq \mathcal{V}$ where $\mathcal{V}_2 \subseteq \mathcal{V}_3$.

For such functions, it has been shown that a greedy algorithm achieves near-optimality (Krause and Guestrin, 2007; Krause et al., 2008).

Problem 2 (KNOWN-CON: known world map; constrained travel cost)**.** *Given a world map ϕ, a time horizon T and a travel cost budget B, find a path ξ that maximizes utility*

$$\begin{aligned} \arg\max_{\xi \in \Xi} \quad & \mathcal{F}(\xi, \phi) \\ s.t. \quad & \mathcal{T}(\xi, \phi) \leq B \\ & |\xi| \leq T + 1 \end{aligned} \tag{2}$$

Problem 2 introduces a routing constraint (due to $\mathcal{T}$) for which greedy approaches can perform arbitrarily poorly. Such problems occur when a physical system has to travel between nodes. Chekuri and Pal (2005) and Singh et al. (2007) proposed a quasi-polynomial-time recursive greedy approach to solving this problem. Iyer and Bilmes (2013) solved a related problem (submodular knapsack constraints) using an iterative greedy approach that was generalized by Zhang and Vorobeychik (2016). Yu et al. (2014) proposed a mixed integer approach to solve a related correlated orienteering problem. Hollinger and Sukhatme (2013) proposed a sampling-based approach. Arora and Scherer (2017) used an efficient traveling salesman problem (TSP) with a random sampling approach.

2.1.3. Problems with hidden world maps. We now consider the setting where the world map ϕ is hidden. Given a prior distribution $P(\phi)$, it can be inferred only via the measurements y_i received as the robot visits nodes v_i. Hence, instead of solving for a fixed path, we compute a policy that maps history of measurements received and nodes visited to decide which node to visit.

Problem 3 (HIDDEN-UNC: hidden world map; unconstrained travel cost)**.** *Given a distribution of world maps, $P(\phi)$, a fully connected graph $\mathcal{G}$, a time horizon T, find a policy that at time t, maps the history of nodes visited $\{v_i\}_{i=1}^t$ and measurements received $\{y_i\}_{i=1}^t$ to compute the next node v_{t+1} to visit at time $t+1$, such that the expected utility is maximized.*

Such a problem occurs for sensor placement where sensors can optionally fail (Golovin and Krause, 2011). Owing to the hidden world map ϕ, it is not straightforward to apply the approaches of Problem KNOWN-UNC: we have to reason both about $P(\phi \mid \{v_i\}_{i=1}^t, \{y_i\}_{i=1}^t)$ and how the function will evolve. However, in some instances the utility function $\mathcal{F}$ has an additional property of *adaptive submodularity* (Golovin and Krause, 2011). This is an extension of the submodularity property where the gain of the function is measured in expectation over the conditional distribution over world maps $P(\phi \mid \{v_i\}_{i=1}^t, \{y_i\}_{i=1}^t)$. Under such situations, applying greedy strategies to Problem 3 has near-optimality guarantees (Chen et al., 2016b, 2015; Golovin et al., 2010; Javdani et al., 2014, 2013). However, these strategies require explicitly sampling from the posterior distribution over ϕ, which makes it intractable to apply for our setting.

Problem 4 (HIDDEN-CON: hidden world map; constrained travel cost)**.** *Given a distribution of world maps, $P(\phi)$, a time horizon T, and a travel cost budget B, find a policy that at time t, maps the history of nodes visited $\{v_i\}_{i=1}^t$ and measurements received $\{y_i\}_{i=1}^t$ to compute the next node v_{t+1} to visit at time $t+1$, such that the expected utility is maximized.*

Such problems crop up in a wide number of areas such as sensor planning for 3D surface reconstruction (Isler et al., 2016) and indoor mapping with UAVs (Charrow et al., 2015; Nelson and Michael, 2015). Problem 4 cannot be solved efficiently by exploiting the adaptive submodularity property owing to the introduction of travel constraints. Hollinger et al. (2013, 2017) proposed a heuristic-based approach to select a subset of informative nodes and perform minimum cost tours. Singh et al. (2009) replanned every step using a non-adaptive information path-planning algorithm. Inspired by adaptive TSP approaches by Gupta et al. (2010), Lim et al. (2015, 2016) proposed recursive coverage algorithms to learn policy trees. However, such methods cannot scale well to large state and observation spaces. Heng et al. (2015) made a modular approximation of the objective function. Isler et al. (2016) surveyed a broad number of myopic information gain-based heuristics that work well in practice but have no formal guarantees.

2.2. Search-based planning

We now present a framework for search-based planning where the objective is to find a feasible path from start to goal while minimizing search effort. We use this framework to pose the problem of learning the optimal heuristic for a given distribution over worlds and briefly discuss prior work on this topic.

2.2.1. Framework. We consider the problem of search on a graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where vertices $\mathcal{V}$ represent robot configurations and edges $\mathcal{E}$ represent potentially valid movements of the robot between these configurations. Given a pair of start and goal vertices, $(v_s, v_g) \in \mathcal{V}$, the objective is to compute a path $\xi \subseteq \mathcal{E}$ - a connected sequence of valid edges. The implicit graph $\mathcal{G}$ can be compactly represented by (v_s, v_g) and a successor function $\mathsf{Succ}(v)$ which

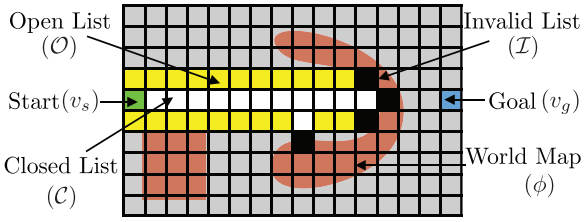


Fig. 3. The search-based planning problem. Given a world map ϕ, the agent has to guide a search tree from start v_s to goal v_g by expanding vertices. At any given iteration, the open list $\mathcal{O}$ represents the set of candidate vertices that can be expanded. The closed list $\mathcal{C}$ represents the set of vertices already expanded. The invalid list represents the set of edges that were found to be in collision with the world. The status of every other vertex is unknown. The search continues until the goal belongs to the open list, i.e. a feasible path to goal has been found.

returns a list of outgoing edges and child vertices for a vertex $v \in \mathcal{V}$. Hence a graph $\mathcal{G}$ 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 $e \in \mathcal{E}$ 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 (Dellin and Srinivasa, 2016).

Algorithm 1 defines a general search based planning algorithm `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 $\mathcal{O} \subset \mathcal{V}$ of candidate vertices to be expanded and a closed list $\mathcal{C} \subset \mathcal{V}$ of vertices which have already been expanded. It also retains an additional invalid list $\mathcal{I} \subset \mathcal{E}$ 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 $\mathcal{V}_{\text{succ}}$ as well as invalid edges $\mathcal{E}_{\text{inv}}$. The lists are then updated and the process repeated till the goal vertex v_g is uncovered. Figure 3 illustrates this framework.

2.2.2. The optimal heuristic problem. In this work, we focus on the *feasible path problem* and ignore the optimality of the path. Although this is a restrictive setting, quickly finding the feasible path is a very important problem in robotics. Efficient feasible path planners such as RRT-Connect (Kuffner and LaValle, 2000) have been proven to be highly effective in high-dimensional motion-planning applications[1] such as robotic arm planning (LaValle, 2006) and mobile robot planning (Laumond et al., 1998). Hence, we ignore the traversal cost of an edge and deal with unweighted graphs. We defer discussions on how to relax this restriction to Section 8.2.

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

1: $\mathcal{O} \leftarrow v_s$, $\mathcal{C} \leftarrow \emptyset$, $\mathcal{I} \leftarrow \emptyset$
2: **while** $\nexists \texttt{Path}(v_s, v_g)$ **do**
3: $\quad v \leftarrow \texttt{Select}(\mathcal{O})$
4: $\quad (\mathcal{V}_{\text{succ}}, \mathcal{E}_{\text{inv}}) \leftarrow \texttt{Expand}(v, \texttt{Succ}, \texttt{Eval}, \phi)$
5: $\quad \mathcal{O} \leftarrow \mathcal{O} \cup \mathcal{V}_{\text{succ}}$, $\mathcal{C} \leftarrow \mathcal{C} \cup v$, $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{E}_{\text{inv}}$
6: **Return** $\texttt{Path}(v_s, v_g)$

We view a heuristic policy as a *selection function* (Algorithm 1, Line 3) that selects a vertex v from the open list $\mathcal{O}$. The objective of the policy is to minimize the number of expansions until the search terminates. Note that the evolution of the open list $\mathcal{O}$ depends on the underlying world map ϕ which is hidden. Given a prior distribution over world maps $P(\phi)$, the true world map can be inferred only via the outcome of the expansion operation $(\mathcal{V}_{\text{succ}}, \mathcal{E}_{\text{inv}})$. The history of outcomes is captured by the state of the search, i.e. the combination of the three lists $\{\mathcal{O}, \mathcal{C}, \mathcal{I}\}$. This is similar to the objective of BUGSY (Ruml and Do, 2007), which defines an utility function for best first search to select nodes to expand by balancing solution cost versus expansion time.

Problem 5 (OPT-HEUR)**.** *Given a distribution of world maps, $P(\phi)$, find a heuristic policy that at time t, maps the state of the search $\{\mathcal{O}_t, \mathcal{C}_t, \mathcal{I}_t\}$ to select a vertex $v_t \in \mathcal{O}_t$ to expand, such that the expected number of expansions until termination is minimized.*

The problem of heuristic design has a lot of historical significance. A common theme is "optimism under uncertainty." A spectrum of techniques exist to manually design good heuristics by relaxing the problem to obtain guarantees with respect to optimality and search effort (Pearl, 1984). To get practical performance, these heuristics are inflated, as has been the case in the applications in mobile robot planning (Likhachev and Ferguson, 2009). However, being optimistic under uncertainty is not a foolproof approach and could be disastrous in terms of search efforts depending on the environment (see LaValle, 2006: Figure 2.5).

Learning heuristics falls under machine learning for general-purpose planning (Jiménez et al., 2012). Yoon et al. (2006) proposed using regression to learn residuals over FF-Heuristic (Hoffmann and Nebel, 2001). Xu et al. (2009, 2007, 2010) improved upon this in a beam-search framework. Arfaee et al. (2011) iteratively improve heuristics. ús Virseda et al. (2013) learned combinations of heuristics to estimate cost-to-go. The Kendall rank coefficient was used to learn open list ranking (Garrett et al., 2016; Wilt and Ruml, 2015). Thayer et al. (2011) learned heuristics online during search. Paden et al. (2017) learned admissible heuristics as sum of squares (SOS) problems. However, these methods did not address minimization of search effort and also ignored the non-i.i.d. nature of the problem.

2.3. POMDP

POMDPs (Kaelbling et al., 1998) provide a rich framework for sequential decision making under uncertainty. However, solving a POMDP is often intractable: finite horizon POMDPs are PSPACE-complete (Papadimitriou and Tsitsiklis, 1987) and infinite horizon POMDPs are undecidable (Madani et al., 2003). Despite this challenge, the field has forged on and produced a vast amount of work by investigating effective approximations and analyzing the structure of the optimal solution. We refer the reader to Ross et al. (2008) for a concise survey of modern approaches.

There are two main approaches to POMDP planning: offline policy computation and online search. In offline planning, the agent computes beforehand a policy by considering all possible scenarios and executes the policy based on the observation received. Although offline methods have shown success in planning near-optimal policies in several domains (Kurniawati et al., 2008; Smith and Simmons, 2012; Spaan and Vlassis, 2005), they are difficult to scale up due to the exponential number of future scenarios that must be considered.

Online methods interleave planning and execution. The agent plans with the current belief, executes the action and updates the belief. Monte Carlo sampling methods explicitly maintain probability over states and plan via Monte Carlo roll-outs (Asmuth and Littman, 2011; McAllester and Singh, 1999). This limits scalability since belief update can take time. In contrast, partially observable Monte Carlo problems (POMCPs) (Silver and Veness, 2010) maintains a set of particles to represent belief and employ Upper Confidence Tree (UCT) methods to plan with these particles. This allows the method to scale up for larger state spaces.

However, the disadvantage of purely online methods is that they require a lot of search effort online and can lead to poor performance due to evaluation on a small number of particles. Somani et al. (2013) presented a state-of-the-art algorithm DESPOT that combines the best aspects of many algorithms. First, it uses determinized sampling techniques to ensure that the branching factor of the tree is bounded (Kearns et al., 2000; Ng and Jordan, 2000). Second, it uses offline precomputed policies to roll-out from a vertex, thus lower bounding its value. Finally, it tries to regularize the search by weighing the utility of a node to be robust against the fact that a finite number of samples is being used.

The methods we have talked about explicitly models the belief. For large-scale POMDPs, this might be an issue. Model-free approaches and representation learning offer attractive alternatives. Model-free policy improvement has been successfully used to solve POMDPs (Li et al., 2009; Liu et al., 2013). Predictive state representations (Boots et al., 2011; Littman and Sutton, 2002) that minimize prediction loss of future observations offer more compact representations than maintaining belief. There also has been a lot of success in employing deep learning to learn powerful representations Hausknecht and Stone (2015); Karkus et al. (2017).

2.4. Reinforcement learning and IL

Reinforcement Learning (RL) (Sutton and Barto, 1998) especially deep RL has dramatically advanced the capabilities of sequential decision making in high-dimensional spaces such as controls Duan et al. (2016), video games Silver et al. (2016), and strategy games (Silver et al., 2016). Several conventional supervised learning tasks are now being solved using deep RL to achieve higher performance (Li et al., 2016; Ranzato et al., 2015). In sequential decision making, the prediction of a learner is dependent on the history of previous outcomes. Deep RL algorithms are able to train such predictors by reasoning about the future accumulated cost in a principle manner.

We refer the reader to Kober et al. (2013) for a concise survey on RL and to Arulkumaran et al. (2017) for a survey on deep RL. Training such policies can be classified into two approaches: either *value function-based approach*, where a value function for an action is learnt, or *policy search*, where a policy is directly learnt. The value function methods can themselves be categorized in two categories, *model-free* algorithms and *model-based* algorithms.

Model-free methods are computationally cheap but ignore the dynamics of the world thus requiring a lot of samples. Q-learning (Watkins and Dayan, 1992) is a representative algorithm for estimating the long-term expected return for executing an action from a given state. When the number of state action pairs are too large in number to track each uniquely, a function approximator is required to estimate the value. Deep Q-learning (Mnih et al., 2015; Wang et al., 2016) addresses such a need by employing a neural-network as a function approximator and learning these network weights. However, the process of using the same network to generate both target values and update Q-values results in oscillations. Hence, a number of remedies are required to maintain stability such as having a buffer of experience, a separate target network and an adaptive learning rate. These are indicative of the underlying sample inefficiency problem of a model-free approach.

Model-based methods such as R-Max (Brafman and Tennenholtz, 2002) learn a model of the world that is then used to plan for actions. While such methods are sample efficient, they require a lot of exploration to learn the model. Even in the case when the model of the environment is known, solving for the optimal policy might be computationally expensive for large spaces. Policy search approaches are commonly used where its easier to parameterize a policy than learn a value function (Peters and Schaal, 2006), however such approaches are sensitive to initialization and can lead to poor local minima.

In contrast with RL methods, IL algorithms (Chang et al., 2015; Daumé et al., 2009; Ross and Bagnell, 2014; Venkatraman et al., 2014) reduce the sequential prediction problem to supervised learning by leveraging the fact that, for many tasks, at training time we usually have a (near-)optimal cost-to-go oracle. This oracle can either come from a human expert guiding the robot (Abbeel and Ng,

2004) or from ground truth data as in natural language processing (Chang et al., 2015). The existence of such oracles can be exploited to alleviate learning by trial and error: imitation of an oracle can significantly speed up learning. A traditional approach to using such oracles is to learn a policy or value function from a pre-collected dataset of oracle demonstrations (Finn et al., 2016; Ratliff et al., 2009; Ziebart et al., 2008). A problem with these methods is that they require training and test data to be sampled from the same distribution that is difficult in practice. In contrast, interactive approaches to data collection and training has been shown to overcome stability issues and works well empirically (Ross and Bagnell, 2014; Ross et al., 2011; Sun et al., 2017). Furthermore, these approaches lead to strong performance through a reduction to no-regret online learning.

Recent approaches have also employed imitation of clairvoyant oracles, that has access to more information than the learner during training, to improve RL as they offer better sample efficiency and safety. Kahn et al. (2017); Zhang et al. (2016) train policies that map current observation to action by extending guided policy search (Levine and Koltun, 2013) for imitation of model predictive control (MPC) oracles. Tamar et al. (2016) considered a cost-shaping approach for short horizon MPC by offline imitation of long horizon MPC, which is closest to our work. Gupta et al. (2017) developed a holistic mapping and planner framework trained using feedback from optimal plans on a graph.

Sun et al. (2017) also theoretically analyzed the question of why IL aids in RL. They developed a comprehensive theoretical study of IL on discrete MDPs and construct scenarios to show that IL achieves better sample efficiency than any RL algorithm. Concretely, they conclude that one can expect at least a polynomial gap and a possible exponential gap in regret between IL and RL when one has access to unbiased estimates of the optimal policy during training.

3. Problem formulation

In this section, we map the problems we wish to solve in a POMDP setting. We begin with a brief review of POMDP notation in Section 3.1. We then define a mapping from the IPP problem to a POMDP in Section 3.2. We also define a mapping from the search-based planning problem to a POMDP in Section 3.3. Finally, we discuss the hardness of such POMDP problems in Section 3.4.

3.1. POMDPs

A discrete-time finite horizon POMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \Omega, R, \mathcal{O}, Z, T)$ where:

- $\mathcal{S}$ is a set of states;
- $\mathcal{A}$ is a set of actions;
- Ω is a set of state transition probabilities;
- $R : \mathcal{S} \times \mathcal{A}$ is the reward function;
- $\mathcal{O}$ is the set of observations;
- Z is a set of conditional observation probabilities;
- T is the time horizon.

At each time period, the environment is in some state $s \in \mathcal{S}$ that cannot be observed directly. The initial state is sampled from a distribution $P(s)$. The agent takes an action $a \in \mathcal{A}$ that causes the environment to transition to state $s' \in \mathcal{S}$ with probability $\Omega\left(s, a, s'\right) = P(s_{t+1} = s' | s_t = s, a_t = a)$. The agent receives a reward $R(s, a)$. On reaching the new state s', it receives an observation $o \in \mathcal{O}$ according to the probability $Z\left(s', a, o\right) = P(o_{t+1} = o | s_{t+1} = s', a_t = a)$.

A *history* $\psi \in \Psi$ is a sequence of actions and observations $\psi_t = \{\langle o_1 \rangle, \langle a_1, o_2 \rangle, \ldots, \langle a_{t-1}, o_t \rangle\}$. Note that the initial history $\psi_t = \langle o_1 \rangle$ is simply the observation at the initial timestep. The history ψ_t captures all information required to express the belief over state. The belief $P(s_{t+1} | \psi_{t+1})$ can be computed recursively applying Bayes' rule

$$\eta\, Z(s_{t+1}, a_t, o_{t+1}) \sum_{s_t \in \mathcal{S}} \Omega(s_t, a_t, s_{t+1})\, P(s_t | \psi_t)$$

where η is a normalization constant.

The history can then also be used to compute an update $P(\psi_{t+1} | \psi_t, a_t)$:

$$\sum_{s_t \in \mathcal{S}} \sum_{s_{t+1} \in \mathcal{S}} P(s_t | \psi_t)\, \Omega(s_t, a_t, s_{t+1})\, Z(s_{t+1}, a_t, o_{t+1})$$

The agent's action selection behavior can be explained by a policy $\pi(\psi_t) \in \Pi$ that maps history ψ_t to action a_t.

Let the state and history distribution induced by a policy π after t timesteps be $P(s, \psi | \pi, t)$. The value of a policy π is the expected cumulative reward for executing π for T timesteps on the induced state and history distribution

$$J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s_t, \psi_t \sim P(s, \psi | \pi, t)} \left[R(s_t, \pi(\psi_t)) \right] \tag{3}$$

The optimal policy maximizes the expected cumulative reward, i.e. $\pi^* \in \arg\max_{\pi \in \Pi} J(\pi)$.

Given a starting history ψ, let $P(s', \psi' | \psi, \pi, i)$ be the induced state history distribution after i timesteps. The value of executing a policy π for t timesteps from a history ψ is the expected cumulative reward:

$$\tilde{V}_t^{\pi}(\psi) = \sum_{i=1}^{t} \mathbb{E}_{s_i, \psi_i \sim P(s', \psi' | \psi, \pi, i)} \left[R(s_i, \pi(\psi_i)) \right] \tag{4}$$

The state-action value function $\tilde{Q}_t^{\pi}(\psi_t, a_t)$ is defined as the expected sum of one-step-reward and value-to-go:

$$\begin{aligned} \tilde{Q}_t^{\pi}(\psi, a) = & \mathbb{E}_{s \sim P(s|\psi)} \left[R(s, a) \right] + \\ & \mathbb{E}_{\psi' \sim P(\psi' | \psi, a)} \left[\tilde{V}_{t-1}^{\pi}(\psi') \right] \end{aligned} \tag{5}$$

3.2. Mapping IPP to POMDPs

We now map IPP problems HIDDEN-UNC and HIDDEN-CON to a POMDP. The state is defined to contain all information that is required to define the reward, observation, and transition functions. Let the state be the set of nodes visited and the underlying world, $s_t = \{v_1, \ldots, v_t, \phi\}$. At the start of an episode, a world is sampled from a prior distribution $\phi \sim P(\phi)$ along with a graph $\mathcal{G} \sim P(\mathcal{G})$. The initial state is assigned by setting $s_1 = \{v_1, \phi\}$. Note that the state s_t is partially observable due to the hidden world map ϕ.

We define the action $a_t = v_{t+1}$ to be the next node to visit. We are now ready to map the utility and travel cost to the reward function definition. Given the agent is in state s_t and has executed a_t, we can extract the path $\xi = (v_1, v_2, \ldots, v_{t+1})$ and the underlying world ϕ. Hence, we can compute the utility function $\mathcal{F}(\xi, \phi)$. We can also compute the travel cost function $\mathcal{T}(\xi, \phi)$.

Before we define the reward function, we note that for Problem HIDDEN-CON, not all actions are feasible at all times due to connectivity of the graph and constraints due to travel cost. Hence, we can define a feasible set of actions $\mathcal{A}_{\text{feas}}(s) \subset \mathcal{A}$ for a state as follows

$$\mathcal{A}_{\text{feas}}(s) = \{a \mid a \in \mathcal{A}, (v_t, v_{t+1}) \in \mathcal{E}, \mathcal{T}(\xi, \phi) \leq B\} \quad (6)$$

For Problem HIDDEN-UNC, let $\mathcal{A}_{\text{feas}}(s) = \mathcal{A}$.

As the objective is to maximize the cumulative reward function, we define the reward to be proportional to the marginal utility of visiting a node. Given a node $v \in \mathcal{V}$, a path ξ and world ϕ, the marginal gain of the utility function $\mathcal{F}$ is $\Delta_{\mathcal{F}}(v \mid \xi, \phi) = \mathcal{F}(\xi \cup \{v\}, \phi) - \mathcal{F}(\xi, \phi)$. The one-step-reward function, $R(s, a)$, is defined as the marginal gain of the utility function. In addition, the reward is set to $-\infty$ whenever an infeasible action is selected. Hence,

$$R(s, a) = . \begin{cases} \Delta_{\mathcal{F}}(a \mid \xi, \phi) & \text{if } a \in \mathcal{A}_{\text{feas}}(s) \\ -\infty & \text{otherwise} \end{cases} \quad (7)$$

The state transition function, $\Omega(s, a, s')$, is defined as the deterministic function which sets $v_{t+1} = a_t$. We define the observation to be the measurement $o_t = y_t$ and the observation model Z to be a deterministic function $o_t = \mathcal{H}(v_t, \phi)$.

Note that the history ψ_t, the sequence of actions and observations, is captured in the sequence of nodes visited $\{v_i\}_{i=1}^t$ and measurements received $\{y_i\}_{i=1}^t$. In our implementation, we encode this information in an occupancy map as described later in Section 6.1. The information-gathering policy $\pi(\psi_t)$ maps this history to an action a_t, the sensing location to visit.

3.3. Mapping search-based planning to POMDPs

We now map the problem of computing a heuristic policy to a POMDP setting. Let the state be the open list and the underlying world, $s_t = \{\mathcal{O}_t, \phi\}$. At the start of an episode, a world is sampled from a prior distribution $\phi \sim P(\phi)$ along with a start state v_s. The initial state is assigned by setting $s_1 = \{v_s, \phi\}$. Note that the state s_t is partially observable due to the hidden world map ϕ.

We define the action a_t as the vertex $v \in \mathcal{O}_t$ that is to be expanded by the search. The state transition function, $\Omega(s, a, s')$, is defined as the deterministic function which sets $\mathcal{O}_{t+1}$ by querying $\texttt{Expand}(v, \texttt{Succ}, \texttt{Eval}, \phi)$. The one-step-reward function, $R(s, a)$, is defined as -1 for every (s_t, a_t) until the goal is added to the open list. In addition, the reward is set to $-\infty$ whenever an infeasible action is selected. Hence,

$$R(s, a) = \begin{cases} -\infty & \text{if } a \notin \mathcal{O} \\ 0 & \text{if } v_g \in \mathcal{O} \\ -1 & \text{otherwise} \end{cases} \quad (8)$$

We define the observation to be the successor nodes and invalid edges, i.e. $o_t = \{\mathcal{V}_{\text{succ}}, \mathcal{E}_{\text{inv}}\}$ and the observation model Z to be a deterministic function $(\mathcal{V}_{\text{succ}}, \mathcal{E}_{\text{inv}}) = \texttt{Expand}(v, \texttt{Succ}, \texttt{Eval}, \phi)$.

Note that the history, the sequence of actions and observations, is contained in the information present in the concatenation of all lists, i.e. $\psi_t = \{\mathcal{O}, \mathcal{C}, \mathcal{I}\}$. The heuristic is a policy $\pi(\psi_t)$ that maps this history to an action a_t, the vertex to expand.

Note that it is more natural to think of this problem as minimizing a one-step-cost than maximizing a reward. Hence, when we subsequently refer to this problem instance, we refer to the cost $c(s, a) = -R(s, a)$ and the cost-to-go $\tilde{Q}_t^{\pi}(\psi, a)$. This only results in a change from maximization to minimization.

3.4. What makes these POMDPs intractable?

A natural question is to ask whether these problems can be solved by state-of-the-art POMDP solvers such as POMCP (Silver and Veness, 2010) or DESPOT (Somani et al., 2013). Although such solvers are very effective at scaling up and solving large-scale POMDPs, there are a few reasons why there are not immediately applicable to our problem.

First, these methods require a lot of online effort. In the case of search-based planning, the effort required to plan in belief space defeats the purpose of a heuristic all together. In the case of IPP, the observation space is very large and belief updates would be time consuming.

Second, since both methods employ a particle-filter-based approach to tracking plausible world maps, they both are susceptible to a realizability problem. It is unlikely that there will be a world map particle that will explain all observations. That being said, the world maps can explain local correlations in observations. For example, when planning indoors the world maps can explain correlations in observations made at intersection of corridors. Hence, we would like to generalize across these local submaps.

4. Imitation of clairvoyant oracles

A possible approach is to employ model free Q-learning (Mnih et al., 2015) by featurizing the history ψ_t and collecting on-policy data. However, given the size of Ψ, this may require a large number of samples. Another strategy is to parameterize the policy class and employ policy improvement (Peters and Schaal, 2006) techniques. However, such techniques when applied to POMDP settings may lead to poor local minima due to poor initialization. We discussed in Section 2.4 how IL offers a more effective strategy than RL in scenarios where there exist good 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. In this section, we extend this principle and show how imitation of *clairvoyant oracles* enables efficient learning of POMDP policies.

We begin in Section 4.1 with a recap of IL which is a non-i.i.d. learning problem of minimizing an imitation loss on the distribution of states encountered by the learner. We then show how IL can be applied in the POMDP framework to compute an approximate solution in Section 4.2. Specifically, we train policies by imitating clairvoyant oracles: oracles that have full access to the state to compute optimal decisions. Finally, in Section 4.3 we analyze the implications of imitating such oracles, which provides insight on when such approximations are effective and when they fail.

4.1. Imitation learning

We now formally define IL as applied to our setting. Given a policy π, we define the distribution of histories $P(\psi|\pi)$ induced by it (termed as *roll-in*). Let $\mathcal{L}(\psi, \pi)$ be a loss function that captures how well policy π imitates an oracle. Our goal is to find a policy $\hat{\pi}$ that minimizes the expected loss as follows:

$$\hat{\pi} = \arg\min_{\pi \in \Pi} \mathbb{E}_{\psi \sim P(\psi|\pi)} [\mathcal{L}(\psi, \pi)] \tag{9}$$

This is a non-i.i.d. supervised learning problem. Ross et al. (2011) proposed FORWARDTRAINING to train a non-stationary policy (one policy $\hat{\pi}_t$ for each timestep), where each policy $\hat{\pi}_t$ can be trained on distributions induced by previous policies $(\hat{\pi}_1, \ldots, \hat{\pi}_{t-1})$. Although this solves the problem exactly, it is impractical given a different policy is needed for each timestep. For training a single policy, Ross et al. (2011) showed how such problems can be reduced to no-regret online learning using dataset aggregation (DAGGER). The loss function they consider $\mathcal{L}$ is a misclassification loss with respect to what the expert demonstrated. Ross and Bagnell (2014) extended the approach to the RL setting where $\mathcal{L}$ is the reward-to-go of an oracle reference policy by aggregating *values* to imitate (AGGREVATE).

4.2. Solving POMDP via imitation of a clairvoyant oracle

To examine the applicability of IL in the POMDP framework, we compare the loss function (9) to the action value function (5). We see that a good candidate loss function $\mathcal{L}(\psi, \pi)$ should incentivize maximization of $\tilde{Q}^{\pi}_{T-t+1}(\psi, \pi(\psi))$. A suitable approximation of the optimal value function $\tilde{Q}^{\pi^*}_{T-t+1}$ that can be computed at train time would suffice. However, we cannot resort to oracles that explicitly reason about the belief over states $P(s_t|\psi_t)$, let alone planning in this belief space due to tractability issues.

In this work, we leverage the fact that for our problem domains, we have access to the true state s_t at train time. This allows us to define oracles that are *clairvoyant*: that can observe the state at training time and plan actions using this information.

Definition 1 (Clairvoyant oracle)**.** *A clairvoyant oracle $\pi_{\mathrm{OR}}(s)$ is a policy that maps state s to action a with an aim to maximize the cumulative reward of the underlying MDP $(\mathcal{S}, \mathcal{A}, \Omega, R, T)$.*

The oracle policy defines an equivalent action value function *defined on the state* as follows:

$$Q^{\pi_{\mathrm{OR}}}_t(s, a) = R(s, a) + \mathbb{E}_{s' \sim P(s'|s,a)} \left[V^{\pi_{\mathrm{OR}}}_{t-1}(s') \right] \tag{10}$$

Our approach is to imitate the oracle during training. This implies that we train a policy $\hat{\pi}$ by solving the following optimization problem

$$\hat{\pi} = \arg\max_{\pi \in \Pi} \mathbb{E}_{\substack{t \sim U(1:T), \\ s_t, \psi_t \sim P(s, \psi|\pi, t)}} \left[Q^{\pi_{\mathrm{OR}}}_{T-t+1}(s_t, \pi(\psi_t)) \right] \tag{11}$$

Although we define training procedures to concretely realize (11) later in Section 5, we offer some intuition behind this approach. As the oracle π_{OR} knows the state s, it has appropriate information to assign a value to an action a. The policy $\hat{\pi}$ attempts to imitate this action from the partial information content present in its history ψ. Owing to this realization error, the policy $\hat{\pi}$ visits a different state, updates the history, and queries the oracle for the best action. Hence, although the learnt policy can make mistakes in the beginning of an episode, with time it gets better at imitating the oracle.

4.3. Analysis using a hallucinating oracle

The learnt policy imitates a clairvoyant oracle that has access to more information (state s compared with history ψ). This results in a large realizability error that is due to two terms: first, the information mismatch between s and ψ, and second, the expressiveness of feature space. This realizability error can be hard to bound making it difficult to apply the performance guarantee analysis of Ross and Bagnell (2014). It is also not desirable to obtain a performance bound with respect to the *clairvoyant oracle* $J(\pi_{\mathrm{OR}})$.

To alleviate the information mismatch, we take an alternate approach to analyzing the learner by introducing a purely hypothetical construct: a *hallucinating oracle*.

Definition 2 (Hallucinating oracle)**.** *A hallucinating oracle $\tilde{\pi}_{\text{OR}}$ computes the instantaneous posterior distribution over state $P(s|\psi)$ and returns the expected clairvoyant oracle action value.*

$$\tilde{Q}^{\tilde{\pi}_{\text{OR}}}_{T-t+1}(\psi, a) = \mathbb{E}_{s\sim P(s|\psi)}\left[Q^{\pi_{\text{OR}}}_{T-t+1}(s, a)\right] \tag{12}$$

We show that by imitating a clairvoyant oracle, the learner effectively imitates the corresponding hallucinating oracle.

Lemma 1. *The **offline** imitation of **clairvoyant** oracle (11) is equivalent to **online** imitation of a **hallucinating** oracle as shown by*

$$\hat{\pi} = \arg\max_{\pi\in\Pi} \mathbb{E}_{\substack{t\sim U(1:T),\\ \psi_t\sim P(\psi|\pi,t)}}\left[\tilde{Q}^{\tilde{\pi}_{\text{OR}}}_{T-t+1}(\psi_t, \pi(\psi_t))\right]$$

Here offline imitation refers to a training procedure (11) where the oracle can access the world map whereas online imitation refers to a training procedure where the map is inaccessible and the hallucinating oracle only requires the history ψ.

Proof. Refer to Appendix 8.4. ■

Note that a hallucinating oracle uses the same information content as the learnt policy. Hence, the realization error is purely due to the expressiveness of the feature space. The empirical risk of imitating the hallucinating oracle will be significantly lower than the risk of imitating the clairvoyant oracle.

Lemma 1 now allows us to express the performance of the learner with respect to a hallucinating oracle. This brings us to the key question: how good is a hallucinating oracle? Upon examining (12) we see that this oracle is equivalent to the well-known QMDP policy first proposed by Littman et al. (1995). The QMDP policy ignores observations and finds the $Q_{\text{MDP}}(s, a)$ values of the underlying MDP. It then estimates the action value by taking an expectation on the current belief over states $P(s|\psi)$. This estimate amounts to assuming that any uncertainty in the agent's current belief state will be gone after the next action. Thus, the action where long-term reward from all states (weighed by the probability) is largest will be the one chosen.

Littman et al. (1995) pointed out that policies based on this approach are remarkably effective. This has been verified by other works such as Koval et al. (2014) and Javdani et al. (2015). This naturally leads to the question of why we cannot directly apply QMDP to our problem. The QMDP approach requires explicitly sampling from the posterior over states online: a step that we cannot tractably compute as discussed in Section 3.4. However, by imitating clairvoyant oracles, we implicitly obtain such a behavior.

Imitation of clairvoyant oracles has been shown to be effective in other domains such as receding horizon control via imitating MPC methods that have full information (Kahn et al., 2017). Sun et al. (2017) showed how the partially observable acrobot can be solved by imitation of oracles having full state. Karkus et al. (2017) introduce imitation of QMDP in a deep learning architecture to train POMDP policies end to end.

The connection with a hallucinating oracle also provides valuable insight into potential failure situations. Littman et al. (1995) pointed out that policies based on this approach will not take actions to gain information. We discuss such situations in Section 8.1.

5. Approach

In this section, we discuss the various training algorithms to compute policies that can be used in both IPP and search-based planning domains. We begin in Section 5.1 with a discussion of the two classes of algorithms solving IL: ForwardTraining when we can learn non-stationary policies and AggreVaTe when we have to learn stationary policies. We discuss performance guarantees and shortcomings for each of the algorithms. We then apply these algorithms on IPP problems in Section 5.2. The key step is the choice of the clairvoyant oracle to imitate. We show that for Hidden-Unc, one can exploit the adaptive submodular structure of the problem to learn near-optimal policies by imitating the clairvoyant one-step-reward. For Hidden-Con, one has to imitate the clairvoyant reward-to-go. In Section 5.3, we apply AggreVaTe for heuristic learning in search based planning. In this case, we imitate the backwards Dijkstra planner that computes the optimal expansions to go for every state.

5.1. Algorithms

We introduced IL and its applicability to POMDPs in Section 4. We now present a set of algorithms to concretely realize the process. The overall idea is as follows: we are training a policy $\hat{\pi}(\psi)$ that maps features extracted from the history ψ to an action a. The training objective is to imitate a clairvoyant oracle that has access to the corresponding full state s. To define concrete algorithms, we need to reason about two classes of policies: non-stationary and stationary.

5.1.1. Non-stationary policy. For the non-stationary case, we have a policy for each timestep $\hat{\pi}^1, \ldots, \hat{\pi}^T$. The motivation for adopting such a policy class is that the problems arising from the non-i.i.d. distribution immediately disappears. Such a policy class can be trained using the ForwardTraining algorithm (Ross et al., 2011), which sequentially trains each policy on the distribution of features induced from the previous set of policies. Hence, the training problem for each policy at timestep t is reduced to supervised learning.

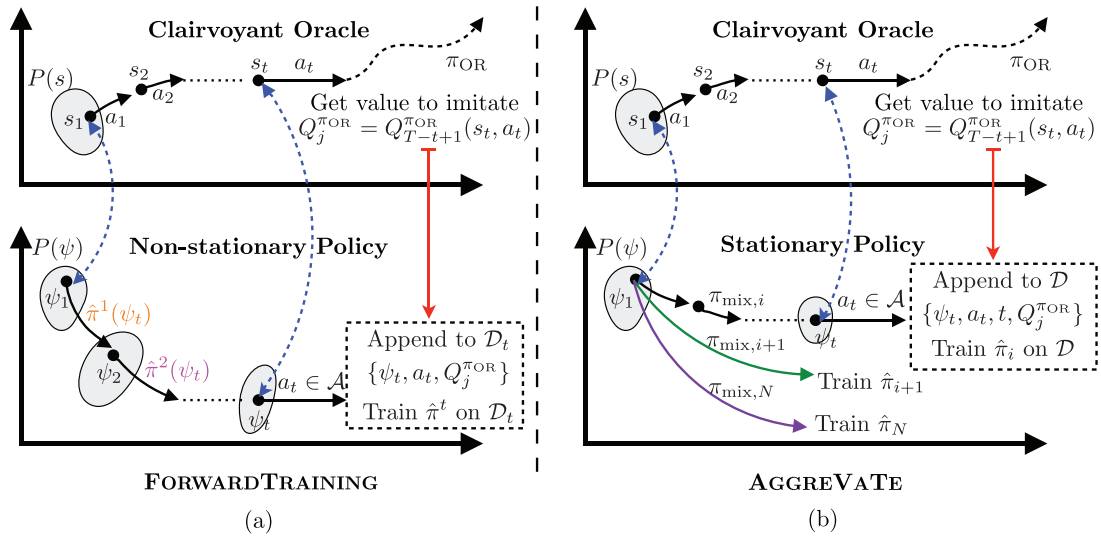


Fig. 4. Overview of the two approaches for training policies. (a) ForwardTraining is used to train a non-stationary policy, i.e. a sequence of policies $\hat{\pi}^1, \ldots, \hat{\pi}^T$ at each timestep. To train a policy at timestep t, a state s is sampled from initial distribution $P(s)$. The policies $\hat{\pi}^1, \ldots, \hat{\pi}^{t-1}$ are then used to roll-in to get (s_t, ψ_t). The oracle is queried to get $Q_{T-t+1}^{\pi_{\text{OR}}}(s_t, a_t)$ which is then used to update the dataset and train policy $\hat{\pi}^t$. (b) AggreVaTe is used to train a stationary policy. The training process is iterative where dataset collection is interleaved with learning. At iteration i, a mixture policy $\pi_{\text{mix},i}$ is used to roll-in to get (s_t, ψ_t). The oracle is queried to get $Q_{T-t+1}^{\pi_{\text{OR}}}(s_t, a_t)$. The data is then aggregated to the whole dataset, which is used to update the entire policy $\hat{\pi}^i$.

Algorithm 2 ForwardTraining (Non-stationary policy)

1: **for** $t = 1$ **to** T **do**
2: Initialize $\mathcal{D}_t \leftarrow \emptyset$.
3: **for** $j = 1$ **to** m **do**
4: Sample initial state s_1 from dataset $P(s)$
5: Execute policy $\hat{\pi}^1, \ldots, \hat{\pi}^{t-1}$ to reach (s_t, ψ_t).
6: Execute any action $a_t \in \mathcal{A}$.
7: Collect value to go $Q_j^{\pi_{\text{OR}}} = Q_{T-t+1}^{\pi_{\text{OR}}}(s_t, a_t)$
8: $\mathcal{D}_t \leftarrow \mathcal{D}_t \cup \{\psi_t, a_t, Q_j^{\pi_{\text{OR}}}\}$
9: Train cost-sensitive classifier $\hat{\pi}^t$ on $\mathcal{D}_t$
10: **Return** Set of policies for each timestep $\hat{\pi}^1, \ldots, \hat{\pi}^T$.

Algorithm 2 describes the ForwardTraining procedure to train the non-stationary policy. The policies are trained in a sequential manner. At each timestep t, the previously trained policies $\hat{\pi}^1, \ldots, \hat{\pi}^{t-1}$ are used to create a dataset of ψ_t by rolling-in (Lines 1–5). For each such datapoint ψ_t, there is a corresponding state s_t. A random action a_t is sampled and the oracle is queried for the cost-to-go $Q_{T-t+1}^{\pi_{\text{OR}}}(s_t, a_t)$ (Line 7). This is then added to the dataset $\mathcal{D}_t$ which is used to train the policy $\hat{\pi}^t$. This is illustrated in Figure 4.

We can state the following property about the training process.

Theorem 1. ForwardTraining *has the following guarantee*

$$J(\hat{\pi}) \geq J(\tilde{\pi}_{\text{OR}}) - 2T\sqrt{|\mathcal{A}|\ \varepsilon_{\text{class}}} + T\varepsilon_{\text{or}}$$

where $\varepsilon_{\text{class}}$ *is the regression error of the learner and* ε_{or} *is the local oracle suboptimality.*

Proof. Refer to Appendix 8.4. ■

However, there are several drawbacks to using a non-stationary policy. First, it is impractical to have a different policy for each timestep as it scales with T. Although this might be a reasonable approach when T is small (e.g. sequence classification problems (Cohen and Carvalho, 2005)), in our applications T can be fairly large. Second, and more importantly, each policy operates on data for only that timestep, thus preventing generalizations across timesteps. Each policy sees only $\frac{\mathcal{D}}{T}$ fraction of the training data. This leads to a high empirical risk.

5.1.2. Stationary policy. A single stationary policy $\hat{\pi}$ enjoys the benefit of learning on data across all timesteps. However, the non-i.i.d. data distribution implies the procedure of data collection and training cannot be decoupled: the learner must be involved in the data-collection process. Ross and Bagnell (2014) show that such policies can be trained by reducing the problem to a no-regret online learning setting. They present an algorithm, AggreVaTe that trains the policy in an interactive fashion where data is collected by a mixture policy of the learner and the oracle, the data is then *aggregated* and the learner is trained on this aggregated data. This process is repeated.

Algorithm 3 describes the AggreVaTe procedure to train the stationary policy. To overcome the non i.i.d distribution issue, the algorithm interleaves data-collection with learning and iteratively trains a set of policies

Algorithm 3 AGGREVATE (Stationary policy)

1: Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π
2: **for** $i = 1$ **to** N **do**
3: Initialize sub-dataset $\mathcal{D}_i \leftarrow \emptyset$
4: Let roll-in policy be $\pi_{\text{mix},i} = \beta_i \pi_{\text{OR}} + (1 - \beta_i)\hat{\pi}_i$
5: Collect m data points as follows:
6: **for** $j = 1$ **to** m **do**
7: Sample initial state s_1 from dataset $P(s)$
8: Sample uniformly $t \in \{1, 2, \ldots, T\}$
9: Execute $\pi_{\text{mix},i}$ up to time $t-1$ to reach (s_t, ψ_t)
10: Execute any action $a_t \in \mathcal{A}$
11: Collect value-to-go $Q_j^{\pi_{\text{OR}}} = Q_{T-t+1}^{\pi_{\text{OR}}}(s_t, a_t)$
12: $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{\psi_t, a_t, t, Q_j^{\pi_{\text{OR}}}\}$
13: Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$
14: Train cost-sensitive classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$
15: **Return** best $\hat{\pi}_i$ on validation

$\left(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N\right)$. Note that these iterations are not to be confused with timesteps: they are simply learning iterations. A policy $\hat{\pi}_i$ is valid for all timesteps. At iteration i, data is collected by rolling-in with a mixture of the learner and the oracle policy (Lines 1–9). The mixing fraction is chosen to be $\beta_i = (1-\alpha)^{i-1}$. Mixing implies flipping a coin with bias β_i and executing the oracle if heads comes up. A random action a_t is sampled and the oracle is queried for the cost-to-go $Q_{T-t+1}^{\pi_{\text{OR}}}(s_t, a_t)$ (Line 11).

The key step is to ensure that *data is aggregated*. The motivation for doing so arises from the fact that we want the learner to do well on the distribution it induces. Ross and Bagnell (2014) showed that this can be posed as the mixture of learners $\left(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N\right)$ doing well on the induced loss sequences $l_i(\pi)$ at every iteration. If we were to treat each iteration as a game in an online adversarial learning setting, this would be equivalent to having bounded regret with respect to the best policy in hindsight on the loss sequence $(l_1, l_2, \ldots, l_N)$. The strategy of dataset aggregation is an instance of follow the leader and, hence, has bounded regret. Hence, data is appended to the original dataset and used to train an updated learner $\hat{\pi}_{i+1}$ (Lines 13–14).

AGGREVATE can be shown to have the following guarantee.

Theorem 2. *There are N iterations of* AGGREVATE *collecting m regression examples per iteration, which guarantees that with probability at least $1 - \delta$*

$$J\left(\hat{\pi}\right) \geq J\left(\tilde{\pi}_{\text{OR}}\right) - 2T\sqrt{|\mathcal{A}|\left(\varepsilon_{\text{class}} + \varepsilon_{\text{reg}} + \mathcal{O}\left(\sqrt{\log 1/\delta / Nm}\right)\right)} - \mathcal{O}\left(\frac{R\, T \log T}{N}\right) + T\varepsilon_{\text{or}}$$

where $\varepsilon_{\text{class}}$ is the empirical regression regret of the best regressor in the regression class on the aggregated dataset, ε_{reg} is the empirical online learning average regret on the sequence of training examples, R is the range of oracle action value, and ε_{or} is the local oracle suboptimality.

Proof. Refer to Appendix 8.4. ■

5.2. Application to IPP

We now consider the applicability of Algorithms 2 and 3 for learning a policy to plan informative paths. We refer to the mapping of the IPP problem to a POMDP defined in Section 3.2. We first need to define a clairvoyant oracle in this context. Recall that the state $s_t = \{v_1, \ldots, v_t, \phi\}$ is the set of nodes visited and the underlying world. A clairvoyant oracle takes a state action pair (s_t, a_t) as input and computes a value. Depending on whether we are solving Problem HIDDEN-UNC or HIDDEN-CON, we explore two different kinds of oracles:

1. *clairvoyant one-step-reward*;
2. *clairvoyant reward-to-go*.

5.2.1. Solving HIDDEN-UNC *by imitating clairvoyant one-step-reward.* We first define a clairvoyant one-step-reward oracle in the IPP framework.

Definition 3 (Clairvoyant one-step-reward)**.** *A clairvoyant one-step-reward returns an action value $Q_t^{\pi_{\text{OR}}}(s, a) = R(s, a)$ that considers only the one-step-reward. In the context of* HIDDEN-UNC*, it uses the world map ϕ, the current path $\{v_1, \ldots, v_t\}$, the next node to visit $v_{t+1} = a_t$ to compute the value $Q^{\text{OR}}(\phi, \{v_1, \ldots, v_t\}, v_{t+1})$ as the marginal gain in utility, i.e.*

$$\Delta_{\mathcal{F}}\left(v_{t+1} \mid \{v_1, \ldots, v_t\}, \phi\right)$$

To motive the use of clairvoyant one-step-reward, we refer to the discussion on the structure of the Problem HIDDEN-UNC in Section 2.1.3. We assume that the utility function is *adaptive monotone submodular*: it has the property of monotonicity and diminishing returns under the belief over world maps. This property implies the following.

1. *Adaptive monotonicity*: The expected value of the utility can only increase on adding a node, i.e.

$$\mathbb{E}_{\phi \sim P(\phi|\psi)}\left[\Delta_{\mathcal{F}}\left(v \mid \mathcal{V}_\psi, \phi\right)\right] \geq 0$$

for all $v \in \mathcal{V}$, where $\psi = \{v_i\}_{i=1}^p, \{y_i\}_{i=1}^p$, and $\mathcal{V}_\psi = \{v_i\}_{i=1}^p$.

2. *Adaptive submodularity*: The expected gain in adding a node diminishes as more nodes are visited, i.e.

$$\mathbb{E}_{\phi \sim P(\phi|\psi)}\left[\Delta_{\mathcal{F}}\left(v \mid \mathcal{V}_\psi, \phi\right)\right] \geq \mathbb{E}_{\phi \sim P(\phi|\psi')}\left[\Delta_{\mathcal{F}}\left(v \mid \mathcal{V}_{\psi'}, \phi\right)\right]$$

for all $v \in \mathcal{V}$, where $\psi \subseteq \psi'$ (history ψ is contained in history ψ').

For such functions, Golovin and Krause (2011) showed that greedily selecting vertices to visit is near-optimal. We

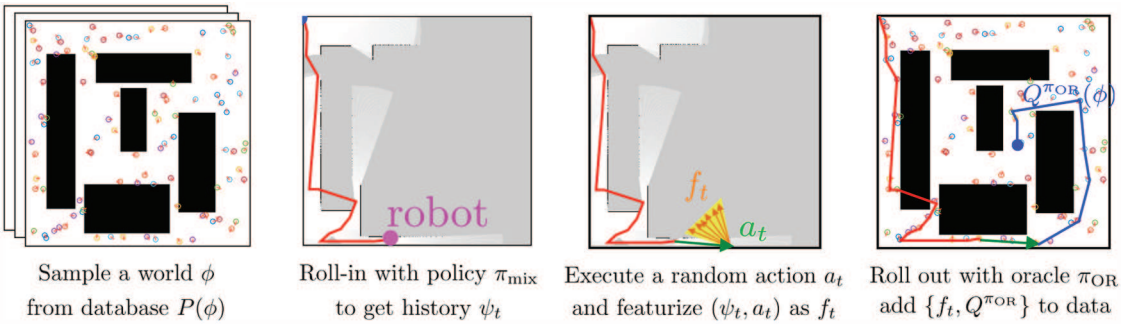


Fig. 5. An overview of QVALAGG in IPP where a learner $\hat{\pi}$ is trained to imitate a clairvoyant oracle π_{OR}. There are 4 key steps. Step 1: A world map ϕ is sampled from database representing $P(\phi)$. Step 2: A mixture policy π_{mix}, of the learner and oracle is used to roll-in on ϕ to a timestep t to get history ψ_t. Step 3: A random action a_t is chosen and (ψ_t, a_t) is featurized as f_t. Step 4: A clairvoyant oracle π_{OR} is given full access to world map ϕ to compute the cumulative reward to go $Q^{\pi_{OR}}$. The pair $(f_t, Q^{\pi_{OR}})$ is added to data to update the learner. This process is repeated to train a sequence of learners.

use this property to show that the clairvoyant one-step-reward induces a one-step-oracle that is equivalent to the greedy policy and, hence, near-optimal. This implies the following theorem.

Theorem 3. *There are N iterations of* AGGREVATE *with Clairvoyant one-step-reward collecting m regression examples per iteration, which guarantees that with probability at least $1-\delta$*

$$J(\hat{\pi}) \geq \left(1-\frac{1}{e}\right) J(\pi^*) - 2T\sqrt{|\mathcal{A}|\left(\varepsilon_{\text{class}} + \varepsilon_{\text{reg}} + \mathcal{O}\left(\sqrt{\log 1/\delta / Nm}\right)\right)} - \mathcal{O}\left(\frac{R\,T\log T}{N}\right)$$

where $\varepsilon_{\text{class}}$ is the empirical regression regret of the best regressor in the regression class on the aggregated dataset, ε_{reg} is the empirical online learning average regret on the sequence of training examples, R is the maximum range of one-step-reward.

Proof. Refer to Appendix 8.4. ■

We show in Section 6 that such policies are remarkably effective. An added benefit of imitating the clairvoyant one-step-reward is that the empirical classification loss $\varepsilon_{\text{class}}$ is lower since only the expected one-step-reward of an action needs to be learnt.

5.2.2. Solving HIDDEN-CON *by imitating clairvoyant reward-to-go.* Unfortunately, Problem HIDDEN-CON does not posses the adaptive-submodular property of HIDDEN-UNC due to the introduction of the travel cost. Hence, imitating the one-step-reward is no longer appropriate. We define the clairvoyant reward-to-go oracle for this problem class.

Definition 4 (Clairvoyant reward-to-go)**.** *A clairvoyant reward-to-go returns an action value $Q_t^{\pi_{OR}}(s,a)$ that corresponds to the cumulative reward obtained by executing a and then following the oracle policy π_{OR}. In the context of* HIDDEN-CON*, it uses the world map ϕ, the current path $\{v_1,\ldots,v_t\}$, the next node to visit $v_{t+1} = a_t$ to solve the problem* KNOWN-CON *and compute a future sequence of nodes $\{v_{t+2},\ldots,v_T\}$. This provides the value $Q^{OR}(\phi,\{v_1,\ldots,v_t\},v_{t+1})$ as the marginal gain*

$$\Delta_{\mathcal{F}}(\{v_{t+1},\ldots,v_T\} \mid \{v_1,\ldots,v_t\},\phi)$$

The corresponding oracle policy π_{OR} is obtained by following the computed path.

Note that solving KNOWN-CON is NP-hard and even the best approximation algorithms require some computation time. Hence the calls to the oracle must be minimized.

Table 1. Mapping from problem and policy type to algorithm

	Problem	
Policy	HIDDEN-UNC	HIDDEN-CON
Non-stationary policy	REWARDFT	QVALFT
Stationary policy	REWARDAGG	QVALAGG

5.2.3. Training and testing procedure. We now present concrete algorithms to realize the training procedure. Given the two axes of variation, problem and policy type, we have four possible algorithms.

1. REWARDFT: Imitate one-step-reward using non-stationary policy by FORWARDTRAINING (Algorithm 2).
2. QVALFT: Imitate reward-to-go using non-stationary policy by FORWARDTRAINING (Algorithm 2).
3. REWARDAGG: Imitate one-step-reward using stationary policy by AGGREVATE (Algorithm 3).
4. QVALAGG: Imitate reward-to-go using stationary policy by AGGREVATE (Algorithm 3).

Table 1 shows the algorithm mapping.

Algorithm 4 QVALAGG

Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π
for $i = 1$ **to** N **do**
  Initialize sub-dataset $\mathcal{D}_i \leftarrow \emptyset$
  Let roll-in policy be $\pi_{\text{mix},i} = \beta_i \pi_{\text{OR}} + (1 - \beta_i)\hat{\pi}_i$
  Collect m data points as follows:
  for $j = 1$ **to** m **do**
    Sample world ϕ from dataset $P(s)$
    Sample start node v_s for $P(v_s)$
    Sample uniformly $t \in \{1, 2, \ldots, T\}$
    Execute $\pi_{\text{mix},i}$ up to time $t-1$ to get path $\{v_1, \ldots, v_t\}$ and history ψ_t
    Sample a random action $a_t \in \mathcal{A}$ as the next vertex to visit $v_{t+1} = a_t$
    Invoke Clairvoyant Reward-to-go oracle to get $Q_j^{\pi_{\text{OR}}} = Q^{\text{OR}}\{\phi, \{v_1, \ldots, v_t\}, v_{t+1}\}$.
    $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{\psi_t, a_t, t, Q_j^{\pi_{\text{OR}}}\}$
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$
  Train cost-sensitive classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$
Return best $\hat{\pi}_i$ on validation

Algorithm 5 SAIL $(P(\phi), P(v_s, v_g), k)$

Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π
for $i = 1$ **to** N **do**
  Initialize sub-dataset $\mathcal{D}_i \leftarrow \emptyset$
  Collect mk data points as follows:
  for $j = 1$ **to** m **do**
    Sample world map $\phi \sim P(\phi)$
    Sample $(v_s, v_g) \sim P(v_s, v_g)$
    Invoke clairvoyant oracle planner to compute $Q^{\pi_{\text{OR}}}(\phi, v)\ \forall\ v \in \mathcal{V}$
    Sample uniformly k timesteps $\{t_1, t_2, \ldots, t_k\}$ where each $t_i \in \{1, \ldots, T\}$
    Roll-out search with $\pi_{\text{mix},i} = \beta_i \pi_{\text{OR}} + (1 - \beta_i)\hat{\pi}_i$
    At each $t \in \{t_1, t_2, \ldots, t_k\}$ pick a random action a_t to get corresponding (ψ_t, v)
    Query oracle for $Q^{\text{OR}}(\phi, a_t)$
    $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{\psi_t, a_t, t, Q^{\text{OR}}(\phi, a_t)\}$
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$
  Train cost-sensitive classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$
Return best $\hat{\pi}_i$ on validation

For completeness, we concretely define the training procedure for QVALAGG in Algorithm 4. The procedure for the remaining three algorithms can be inferred from this. The algorithm iteratively trains a sequence of policies $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$. At every iteration i, the algorithm conducts m episodes. In every episode a different world map ϕ and start vertex (v_s) is sampled from a database. The roll-in is conducted with a mixture policy $\pi_{\text{mix},i}$ that blends the learner's current policy, $\hat{\pi}_{i-1}$, and the oracle's policy, π_{OR}, using a blending parameter β_i. The blending is done in an episodic fashion, with probability β_i the clairvoyant reward-to-go oracle is invoked to compute a path that is followed. With probability $1 - \beta_i$, the learner is invoked for the whole episode. In a given episode, the roll-in is conducted to a timestep t that is uniformly sampled. At the end of the roll-in, we have a path $\{v_1, \ldots, v_t\}$ and a history ψ_t. A random action $a_t \in \mathcal{A}$ is sampled that defines the next vertex to visit $v_{t+1} = a_t$. The clairvoyant reward-to-go oracle is invoked with the world ϕ and the path already traveled $\{v_1, \ldots, v_t\}, v_{t+1}\}$. It then invokes a solver to HIDDEN-CON to complete the path and return the reward to go $Q_j^{\pi_{\text{OR}}}$. This history action pair (ψ_t, a_t) is projected to a feature space along with label $Q_j^{\pi_{\text{OR}}}$. The data is aggregated to the dataset which is eventually used to train policy $\hat{\pi}_{i+1}$. Figure 5 illustrates this approach.

5.3. Application to search-based planning

We now consider the applicability of Algorithm 3 for heuristic learning in search-based planning. Unlike the IPP problem domain, there is no incentive to use a non-stationary policy or imitate clairvoyant one-step-rewards. Hence, we only consider training a stationary policy imitating clairvoyant reward-to-go.

We first need to define a clairvoyant oracle for this problem. Given access to the world map ϕ, the oracle has to solve for the optimal number of expansions to reach the goal. This allows us to define a *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 $v \in \mathcal{V}$ using dynamic programming.

Definition 5 (Clairvoyant oracle planner)**.** *Given full access to the state s, which contains the open list $\mathcal{O}$ and world ϕ, and a goal v_g, the oracle planner encodes the cost-to-go from any vertex $v \in \mathcal{V}$ as the function $Q_t^{\pi_{\text{OR}}}(s, a)$ that implicitly defines an oracle policy, $\pi_{\text{OR}}(s) = \arg\min_{v \in \mathcal{O}} Q_t^{\pi_{\text{OR}}}(s, a)$.*

The clairvoyant oracle planner provides a look-up table $Q^{\text{OR}}(\phi, v)$ for the optimal cost-to-go from any vertex irrespective of the current state of the search.

A key distinction between this oracle and the one defined for an IPP problem in Section 5.2 is that we are able to efficiently 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. In addition, this allows us a better roll-in procedure where the oracle and learner are interleaved. We adapt the AGGREVATE framework to present an algorithm, *search as imitation learning* (SAIL).

Algorithm 5 describes the SAIL framework, which iteratively trains a sequence of policies $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$. For training the learner, we collect a dataset $\mathcal{D}$ as follows: at every iteration i, the agent executed m different searches

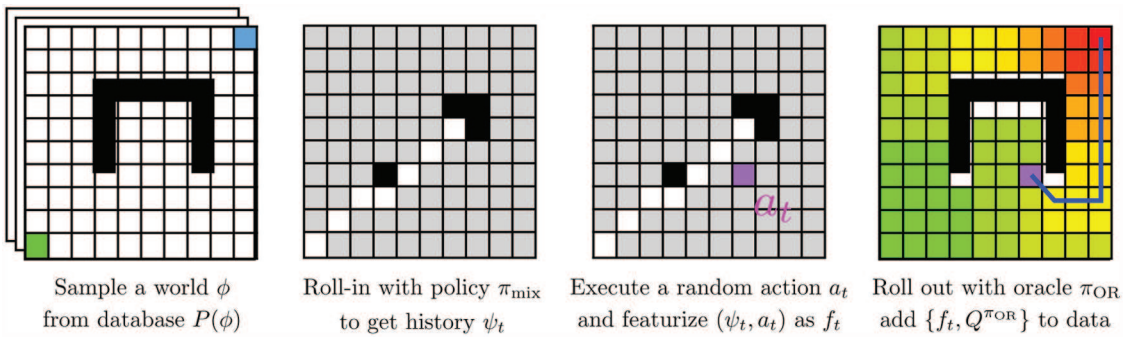


Fig. 6. An overview of SAIL in search-based planning where a learner $\hat{\pi}$ is trained to imitate a clairvoyant oracle π_{OR}. There are four key steps. Step 1: A world map ϕ is sampled from database representing $P(\phi)$ along with start goal pair (v_s, v_g). Step 2: A mixture policy π_{mix}, of the learner and oracle is used to roll-in on ϕ to a timestep t to get history ψ_t, which is the combination of open list, closed list, and invalid edges. Step 3: A random vertex a_t from the open list is chosen and (ψ_t, a_t) is featurized as f_t. Step 4: A clairvoyant oracle π_{OR} is given full access to world map ϕ to compute the cumulative cost to go $Q^{\pi_{\text{OR}}}$. The pair $(f_t, Q^{\pi_{\text{OR}}})$ is added to data to update the learner. This process is repeated to train a sequence of learners.

(Algorithm 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 $\pi_{\text{mix},i}$ that blends the learner's current policy, $\hat{\pi}_i$, and the oracle's policy, π_{OR}, using a 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 $\{\psi_t, a_t, t, Q^{\text{OR}}(\phi, a_t)\}$. The policy $\pi_{\text{mix},i}$ is rolled out until the end of the episode and all the collected data is aggregated with dataset $\mathcal{D}$. 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 $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$. Figure 6 illustrates the SAIL framework.

Note that while the oracle is invoked once per ϕ, we obtain k datapoints: this is critical for speeding up training. We also 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 with uninformed search as discussed further in Appendix 8.4.

6. Experiments on IPP

In this section, we extensively evaluate our approach on a set of 2D and 3D IPP problems across a spectrum of synthetic and real-world environments. We examine a class of IPP problem where a robot, equipped with a range-limited sensor, possibly constrained by time and fuel resources, is tasked with 3D reconstruction of structures in the world. We choose a variety of environments to highlight the importance of adaptive behaviors for information gathering. Our implementation is open source for both MATLAB and C++.[2]

6.1. Problem details

We consider both 2D and 3D IPP problems. The world map ϕ is represented as a 2D or 3D binary grid, i.e. a grid cell is either occupied or free. The candidate set of sensing locations $\mathcal{V}$ is generated by uniformly randomly sampling nodes in the configuration space of the robot. For 2D problems, the configuration space of the robot is $SE(2)$, for 3D problems it is $SE(3)$. We assume for simplicity that the robot can teleport between any two nodes v_i and v_j and the cost of travel is the 2D/3D Euclidean straight-line distance $\mathcal{T}(\{v_i, v_j\}, \phi) = ||v_i - v_j||_2$. It would be straightforward[3] to incorporate practical constraints such as collision avoidance by only allowing motion between vertices that are known to be collision free and computing travel cost to be the arc length distance of a collision-free path.

We assume that the robot is equipped with a field-of-vision (FOV) and range-limited sensor. When a robot visits a node v in a world map ϕ, the measurement received by the robot, $y = \mathcal{H}(v, \phi)$, is computed by ray-casting the sensor on the world and obtaining a scan line (2D) or a depth image (3D).

The utility function $\mathcal{F}$ is selected to be the fractional coverage function (similar to Isler et al. (2016)) that is defined as follows. Let the robot traverse a path $\xi = (v_1, v_2, \ldots, v_p)$ in a world ϕ. For each node $v_i \in \xi$ we have a corresponding measurement y_i. Let the coverage map C_i be a binary grid whose cells are 1 iff the corresponding cell in ϕ is occupied and y_i contains a point in that cell. The total coverage map of a path ξ is a union of all coverage maps $C = \bigcup_{i=1}^{p} C_i$. Then the utility function is the ratio of the total coverage and the total occupied cells in the world map, i.e. $\mathcal{F}(\xi, \phi) = \frac{||C||_1}{||\phi||_1}$.

Although we assume the objective of the robot is to "uncover" every cell of the hidden world map, this framework can also allow a more task-specific objective. For example, if the objective is to perform surface reconstruction of a specific object (and not of every surface in the world map), the utility function can be modified to only cover grid cells belonging to that object. The quality of an observation can also be included in the utility, i.e. measurements at close range can be weighted more than measurements taken from far away.

The values of total timestep T and travel budget B vary with problem instances and are specified along with the results.

The history of events ψ_t is represented as an occupancy grid $\mathcal{X}$ where each grid cell $x \in \mathcal{X}$ corresponds to an occupancy value $P_o(x) \in [0, 1]$. Every time a new measurement is received, $\mathcal{X}$ is updated by ray-casting and applying Bayes' rule (Thrun et al., 2005). The policy $\pi(\psi_t)$ takes as input the occupancy grid and selects an action a_{t+1} that corresponds to the next node v to be visited.

6.2. Baseline: information-theoretic heuristics

Isler et al. (2016) proposed a set of information-theoretic heuristics that quantify the information gain of obtaining a measurement for the task of volumetric reconstruction, which include visibility likelihood and the likelihood of seeing new parts of the object. These heuristics are variants of Shannon's entropy where cells are weighted by an importance function. All of the heuristics are myopic, i.e. given the current occupancy grid, each candidate node is evaluated and the best node is selected as the next action. We briefly describe these heuristics and refer the reader to Isler et al. (2016) for further details.

To evaluate a node v, a set of rays $\mathcal{R}(v)$ are cast from the node using the specifications of the sensor model. A ray $r \in \mathcal{R}$ corresponds to a set of grid cells in the occupancy grid $\mathcal{X}(r)$. Given a grid cell x, the probability of it being occupied is $P_o(x)$ and probability of being free is $\bar{P}_o(x)$. This can be used to compute various information gain metrics according to different heuristics. Let $\mathcal{I}(x)$ be the information stored in the grid cell x. Then the information gain for a node is given by

$$\mathcal{IG}(v) = \sum_{\forall r \in \mathcal{R}(v)} \sum_{\forall x \in \mathcal{X}(r)} \mathcal{I}(x) \tag{13}$$

Depending on the type of information gain $\mathcal{I}$, there can be several information gain functions.

1. Average entropy: $\mathcal{IG}_o(v)$. This corresponds to the entropy

$$\mathcal{I}_o(x) = -P_o(x) \log P_o(x) - \bar{P}_o(x) \log \bar{P}_o(x) \tag{14}$$

2. Occlusion-aware entropy: $\mathcal{IG}_v(v)$. This corresponds to considering the visibility likelihood of a grid cell

$$\mathcal{I}_v(x) = P_v(x)\mathcal{I}_o(x) \tag{15}$$

where $P_v(x)$ is the likelihood of the ray r leading to the x being free.

3. Unobserved voxel: $\mathcal{IG}_u(v)$. This corresponds to only considering unknown grid cells

$$\mathcal{I}_u(x) = \begin{cases} 1 & \text{if } x \text{ is unknown} \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

4. Unobserved entropy: $\mathcal{IG}_k(v)$. This is the composition of unobserved voxel with occlusion-aware entropy

$$\mathcal{I}_k(x) = \mathcal{I}_u(x)\mathcal{I}_v(x) \tag{17}$$

5. Rear side voxel: $\mathcal{IG}_b(v)$. Let RS be the set of *rear-side* grid cells defined as occluded, unknown grid cells adjacent on the ray to an occupied grid cell. Then

$$\mathcal{I}_b(x) = \begin{cases} 1 & \text{if } x \in RS \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

6. Rear side entropy: $\mathcal{IG}_n(v)$. This is the composition of rear side voxel with occlusion-aware entropy

$$\mathcal{I}_n(x) = \mathcal{I}_b(x)\mathcal{I}_v(x) \tag{19}$$

The heuristics are used in a greedy fashion as follows. Given the robot has already visited nodes $v_1, \ldots, v_{t-1}$, it decides to visit node v_t according to the following rule

$$v_t = \arg\max_{v_t \in \mathcal{V}} \frac{\mathcal{IG}(v_t)}{\sum_{v \in \mathcal{V}} \mathcal{IG}(v)} - \lambda \frac{||v_t - v_{t-1}||_2}{\sum_{v \in \mathcal{V}} ||v - v_{t-1}||_2} \tag{20}$$

When applied to the Problem HIDDEN-UNC, we deactivate the penalization and set $\lambda = 0$.

6.3. Imitation learning details

6.3.1. Feature extraction and learner. The policy maps the history ψ to an action a by learning a function approximation for the action value function $\hat{Q}(\psi, a)$. The tuple (a, ψ) is mapped to a vector of features $f = \begin{bmatrix} f_{\text{IG}}^T & f_{\text{mot}}^T \end{bmatrix}^T$. The first set of features $f_{\text{IG}} \in \mathbb{R}^6$ are the information gain heuristics defined in Section 6.2. These heuristics are computed using the occupancy map corresponding to history ψ and the candidate node corresponding to action a. There are several reasons for using these heuristics as the feature vector. They allow generalization across different instance of the world map. They also allow for fair comparison against the heuristics as baseline approaches: the learner learns a trade-off between heuristics.

Here $f_{\text{mot}} \in \mathbb{R}^7$ encodes the distance already traveled by the robot ($\mathbb{R}^1$), the relative translation ($\mathbb{R}^3$) and rotation ($\mathbb{R}^3$) to visit the candidate node from the current node. These set of features capture the travel cost trade-off for visiting a node.

We use random forest regression as a function approximator (Liaw and Wiener, 2002).

6.3.2. Dataset creation. The 2D world maps are created by randomly distributing geometric objects such as rectangles and circles according to a hand-designed parametric distribution. The 3D world maps are created using the ROS-Gazebo simulator and randomly distributing 3D object meshes. Depending on the environment (such as a construction site or office desk), different collections of objects and parametric distributions are selected.

For the experiment on a real dataset, we used registered RGBD data collected by Sturm et al. (2012). The original dataset is a set of registered point clouds along with the measurement pose. This dataset can be used to create the world map ϕ. The set of poses are used to create a fully connected graph $\mathcal{V}$. The algorithm is then restricted to choosing a subset of these poses to maximize the utility. Every time the algorithm visits a node v_i, the corresponding measurement y_i is returned. We found that this setup allowed us to easily evaluate information-gathering algorithms on real data in a completely decoupled manner from the data-collection process.

This process of dataset creation motivates the applicability of our method in practical settings. Given a new environment, we can envision collecting a dataset open-loop, either via manual operation or via some base exploration policy. We can then learn an efficient policy on this dataset and subsequently used the learnt policy for future operations. The generalization capability of the learner allows performance to be transferred to environments with similar object configurations.

6.3.3. Clairvoyant oracle. For algorithms REWARDAGG and REWARDFT, the clairvoyant oracle is simply the one-step-reward function, i.e. the marginal utility of visiting a node given the history of nodes visited. An important implementation detail is that when using the one-step-reward oracle, the call to the oracle is inexpensive. Hence, instead of sampling a random action and obtaining its value, all actions can be queried. This dramatically improves the convergence due to the increase in data size.

For QVALAGG and QVALFT, the clairvoyant oracle needs to solve the submodular routing problem (Problem KNOWN-CON). We use the generalized cost benefit (GCB) (Zhang and Vorobeychik, 2016) algorithm: an efficient greedy algorithm with bi-criterion approximation guarantees. The core idea of the algorithms is very simple: at iteration i select a node v_i that maximizes the ratio of the marginal gain in utility and the marginal gain in travel cost

$$v_i = \arg\max_{v \in \mathcal{V}} \frac{\mathcal{F}\left(v_i \cup \{v_j\}_{j=1}^{i-1}, \phi\right) - \mathcal{F}\left(\{v_j\}_{j=1}^{i-1}, \phi\right)}{\mathcal{T}\left(v_i \cup \{v_j\}_{j=1}^{i-1}, \phi\right) - \mathcal{T}\left(\{v_j\}_{j=1}^{i-1}, \phi\right)} \quad (21)$$

Once a vertex v_i is selected, a TSP solver is invoked to find the minimum cost route through nodes $v_1, \ldots, v_i$ and the vertices are re-ordered accordingly. The process is repeated until the travel budget constraints are met. Note that computing the denominator exactly in (21) might be expensive because it involves a call to a TSP solver. We can instead approximate it by the distance to the node v_i from the last node in the route v_{i-1}.

6.4. Analysis of results

Table 2 shows the utility of all algorithms on various synthetic datasets. The two numbers are lower and upper 95% confidence intervals of the episodic utility of each algorithm. The best performance on each dataset is highlighted. For Problem HIDDEN-UNC, REWARDAGG is employed along with baseline heuristics. For Problem HIDDEN-CON, QVALAGG is employed with baseline heuristic augmented with motion penalization. The train size is 100 and test size is 10. We present a set of observations to interpret these results.

Observation 1. *The learnt policy from* REWARDAGG/QVALAGG *has a consistently competitive performance across all datasets.*

Table 2 shows the performance of all algorithms on a set of 2D and 3D datasets. We see that out of the 10 datasets, the learners perform better than any heuristic on 8. On 2 of the datasets, the Average Entropy heuristic outperforms the learner by a small margin. On examining the datasets, we see that the unknown space exploration behavior of Average Entropy results in good performance in environments that either lack spatial correlation or contain objects distributed in the environment.

Observation 2. *The performance of heuristics vary widely across datasets, however, the performance of the learner is robust.*

We can see that the relative ranking of Average Entropy and Rear Side Voxel interchanges from Dataset 1 to 2. This motivates the need for adaptive policies that assign different utility to unknown cells conditioned on the environment in which the robot is operating. The learner's policy on the other hand adapts to different environments and, hence, maintains a consistently good performance. Interestingly, it also outperforms the heuristic pointwise across datasets, which is indicative of the fact that the adaptation happens during exploration as well.

Observation 3. *The performance margin of* REWARDAGG *in Problem* HIDDEN-UNC *as compared with heuristics is much larger than that of* QVALAGG *in Problem* HIDDEN-CON

This is seen to be especially true in Datasets 1, 2, and 4. As conjectured in Section 5.2.1, this can be attributed to two factors. First, the near-optimality guarantee in Theorem 3 of imitating a clairvoyant one-step-reward bounds the performance of the learner. Second, the empirical regression regret of imitating one step reward values will be much lower than trying to estimate the action values using features from the history ψ_t, i.e. it is easier to predict the immediate utility of going to a sensing location than trying to predict the future utility.

Observation 4. *The performance of Average Entropy in the Poisson Forest dataset is on par with the learner.*

The Poisson Forest dataset is created by sampling circles in the environment from a spatial Poisson distribution where

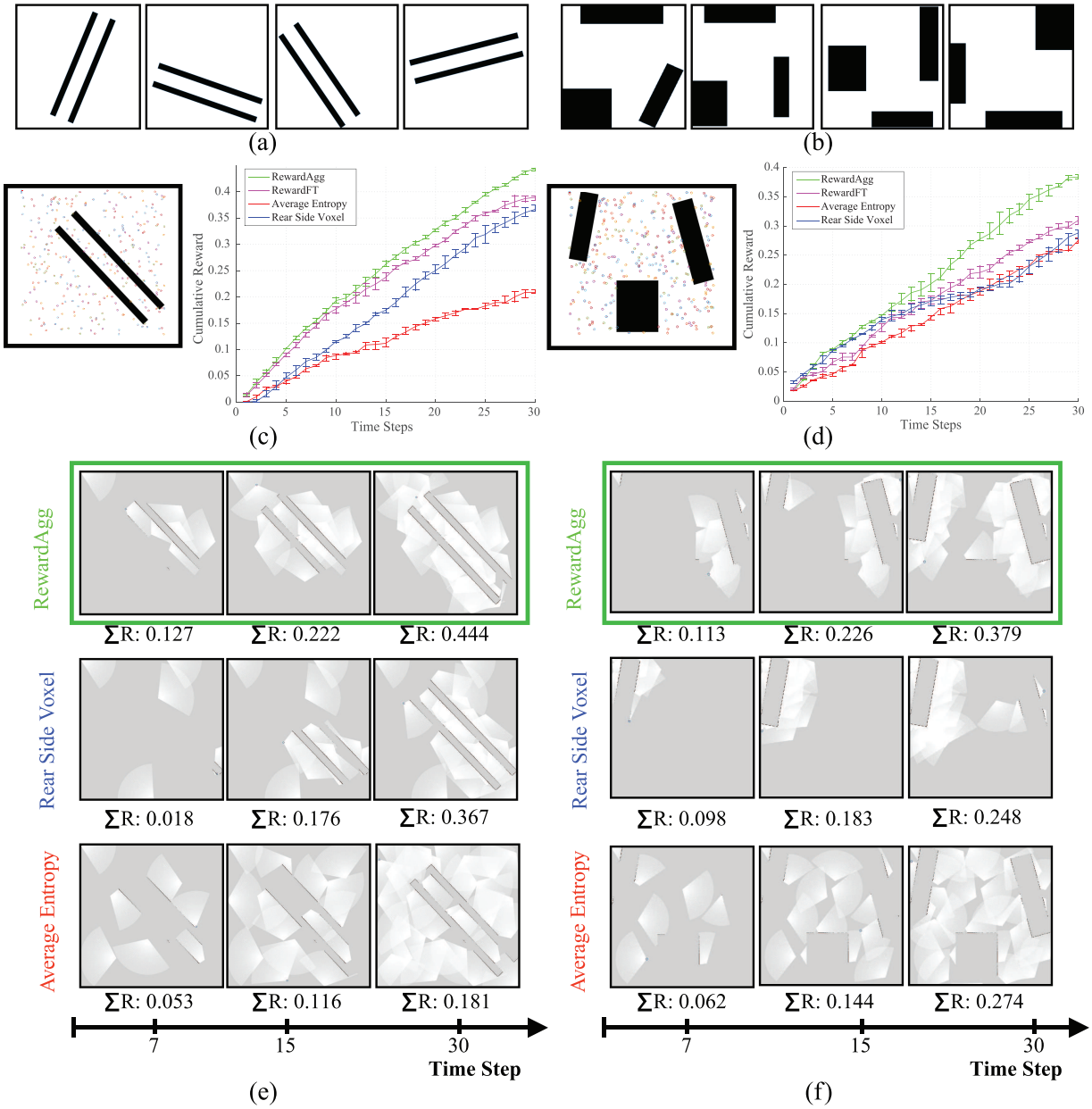


Fig. 7. Case study of Problem HIDDEN-UNC using REWARDAGG, REWARDFT and baseline heuristics. Two different datasets of 2D exploration are considered: (a) dataset 1 (parallel lines) and (b) dataset 2 (distributed blocks). Problem details are: $T = 30$, $|\mathcal{A}| = 300$, 100 training and 100 test maps. A sample test instance is shown along with a plot of cumulative reward with timesteps for different policies is shown in (c) and (d). The error bars show 95% confidence intervals. (e) and (f) Snapshots of the execution at timesteps 7, 15, and 30.

the density of the forest is specified. The lack of spatial correlation, implies it is equally likely to find objects anywhere in the world: an assumption that Average Entropy optimizes for.

6.5. *Case study A: Adaptation to different environments*

We created a set of 2D exploration problems to gain a better understanding of the learnt policies and baseline heuristics. We did this both for Problem HIDDEN-UNC (Figure 7) and HIDDEN-CON (Figure 8). The dataset comprises 2D binary world maps, uniformly distributed nodes, and a simulated laser. The problem details are $T = 30$ and $|\mathcal{A}| = 300$. The cost budget for HIDDEN-CON is $B = 2500$. The training size is 100, test size is 100. REWARDAGG and QVALAGG is executed for 10 iterations.

6.5.1. Dataset 1: Parallel lines. We first examined Problem HIDDEN-UNC. Figure 7(a) shows a dataset created

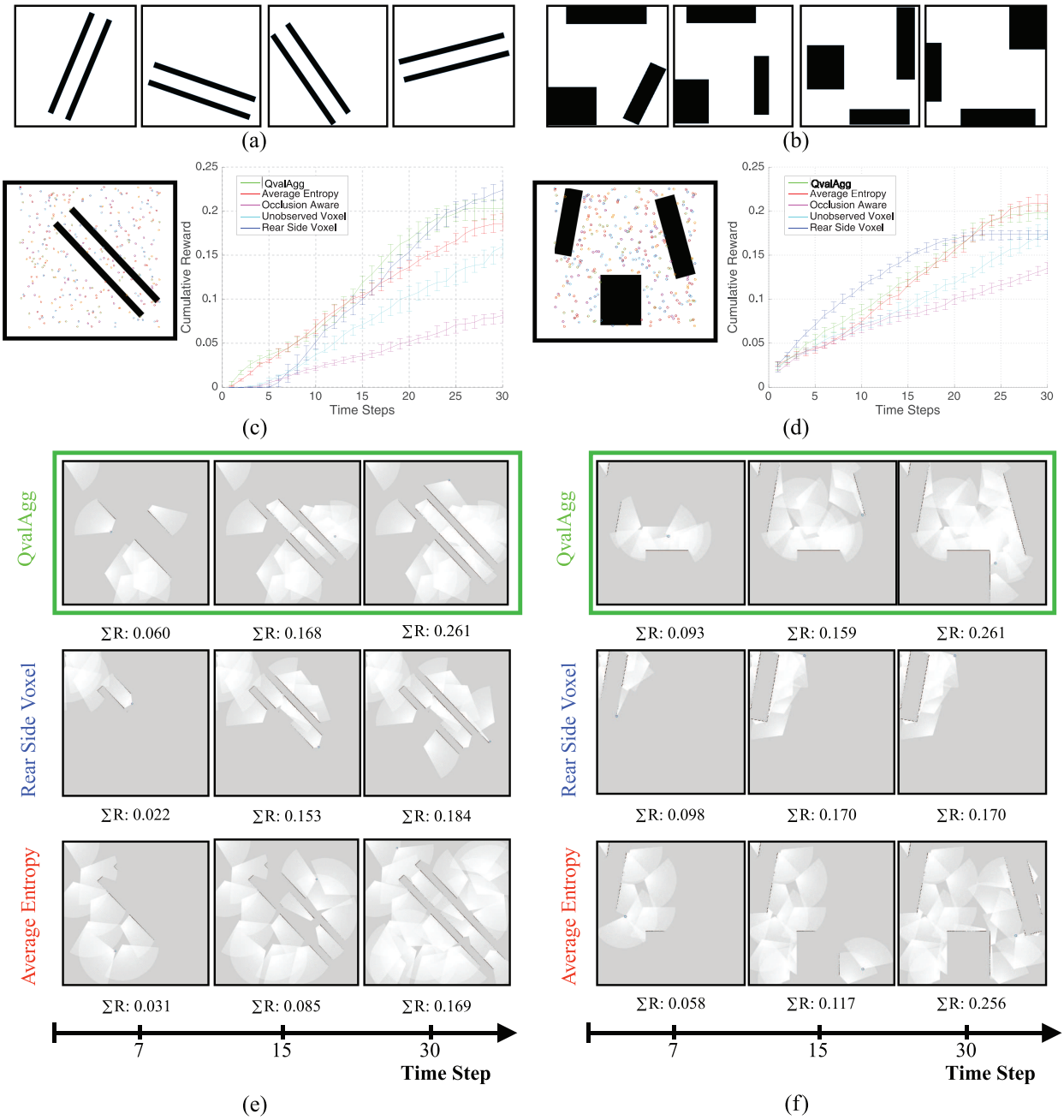


Fig. 8. Case study of Problem HIDDEN-CON usingQVALAGG with baseline heuristics on a 2D exploration problem on two different datasets: dataset 1 (parallel lines) and dataset 2 (distributed blocks). Problem details are: $T = 30$, $B = 2500$, $|\mathcal{A}| = 300$, 100 training and 100 test maps. A sample test instance is shown along with a plot of cumulative reward with timesteps for different policies is shown in (c) and (d) The error bars show 95% confidence intervals Snapshots of execution of QVALAGG, Rear Side Voxel, and Average Entropy are shown for (e) dataset 1 and (f) dataset 2. The snapshots show the evidence grid at timesteps 7, 15, and 30.

by applying random affine transformations to a pair of parallel lines. This dataset is representative of information being concentrated in an area in the environment, e.g. powerline inspection. Figure 7(c) shows a comparison of REWARDAGG, REWARDFT with baseline heuristics. Although Rear Side Voxel outperforms Average Entropy, REWARDAGG outperforms both. Figure 7(e) shows the progress of each. Average Entropy explores the whole world without focusing, Rear Side Voxel exploits early while REWARDAGG trades off exploration and exploitation.

The same trend can be observed in Problem HIDDEN-CON. Figure 8(c) shows a comparison of QVALAGG with baseline heuristics. The heuristic Rear Side Voxel performs the best, while QVALAGG is able to match the heuristic.

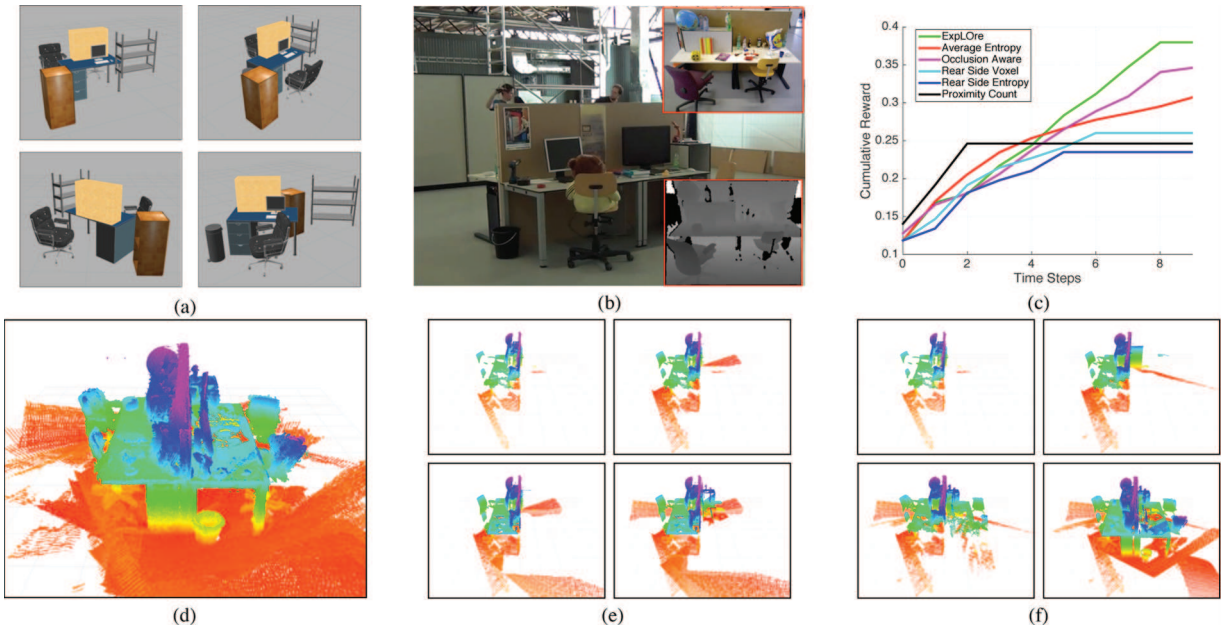


Fig. 9. Comparison of QVALAGG with baseline heuristics on a 3D exploration problem where training is done on simulated world maps and testing is done on a real dataset of an office workspace. The problem details are: $T = 10$, $B = 12$, and $|\mathcal{A}| = 50$. (a) Samples from 100 simulated worlds resembling an office workspace created in Gazebo. (b) Real dataset collected by Sturm et al. (2012) using a RGBD camera. (c) Plot of cumulative reward with timesteps for QVALAGG and baseline heuristics on the real dataset. (d) The 3D model of the real office workspace formed by cumulating measurements from all poses. (e) Snapshots of execution of Occlusion Aware heuristic at timesteps 1, 3, 5, 9. (f) Snapshots of execution of QVALAGG heuristic at timesteps 1, 3, 5, 9.

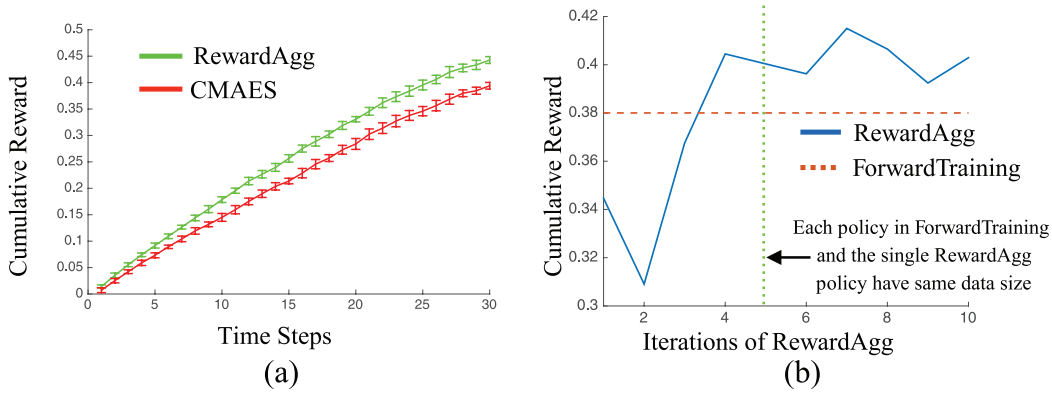


Fig. 10. (a) Comparison of REWARDAGG with CEM policy search. Both algorithms are given access to the same amount of data. The final policy from CEM and the best validation policy of REWARDAGG are then executed on a test dataset. REWARDAGG outperforms CEM not only overall but pointwise at each timestep. (b) Comparison of REWARDAGG with FORWARDTRAINING. Each policy in FORWARDTRAINING is trained with a dataset size of 500. REWARDAGG is trained with 100 samples per iteration for 10 iterations. The performance of both policies on test dataset is shown. REWARDAGG surpasses FORWARDTRAINING at the fourth iteration and never drops below. At iteration 5 the single policy of REWARDAGG has the same dataset size as each policy of the 10 policies of FORWARDTRAINING. However, the single policy still outperforms the non-stationary policy.

Figure 8(e) shows progress of QVALAGG along with two relevant heuristics: Rear Side Voxel and Average Entropy. Rear Side Voxel takes small steps focusing on exploiting viewpoints along the already observed area. Average Entropy aggressively visits the unexplored area which is mainly free space. QVALAGG initially explores the world but on seeing parts of the lines reverts to exploiting the area around it.

6.5.2. Dataset 2: Distributed blocks. We first examined Problem HIDDEN-UNC. Figure 7(b) shows a dataset created by randomly distributing rectangular blocks around

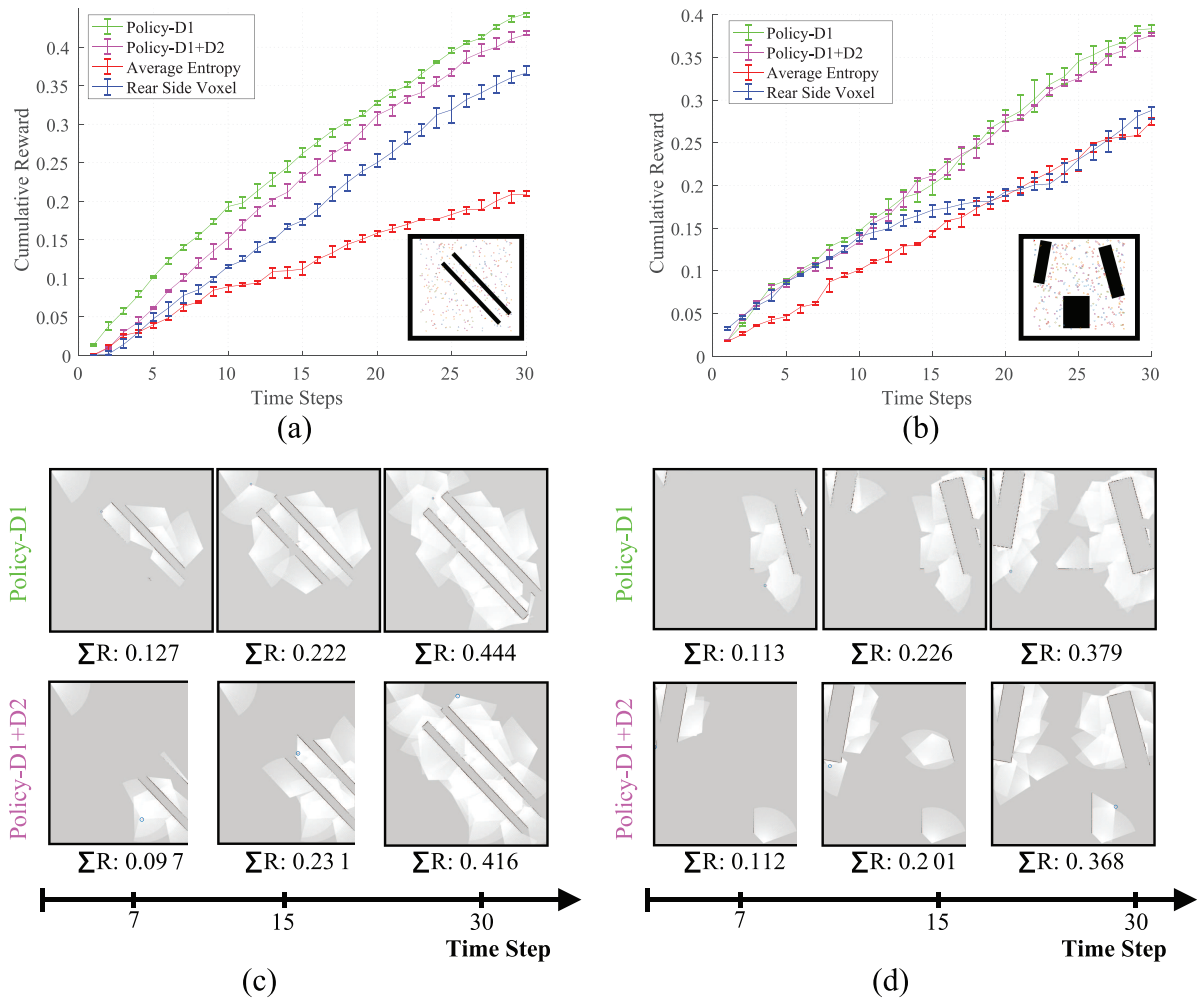


Fig. 11. Comparison of policies trained on homogeneous versus heterogeneous datasets. Policy-D1 is trained on dataset 1 (parallel lines), Policy-D2 on dataset 2 (distributed blocks), and Policy-D1+D2 is trained on both datasets (hence, heterogeneous). (a) Performance of policies on dataset 1. As expected Policy-D1 has the best performance, but is closely followed by Policy-D1+D2. (b) Performance of policies on dataset 2 where a similar trend is observed with respect to Policy-D2. (c) Snapshots at different timestep for each policy executed on a test exampled from dataset 1. Policy-D1+D2 recovers after it uncovers enough about the world. (d) Snapshots on example from dataset 2. Policy-D1+D2 is able to distinguish problem after some initial measurements and match Policy-D2

the periphery of the map. This dataset is representative of information being distributed in the *periphery of the environment, e.g. an empty airplane hangar.* Figure 7(d) shows that Rear Side Voxel saturates early, Average Entropy eventually overtaking it, while REWARDAGG outperforms all. Figure 7(f) shows that Rear Side Voxel gets stuck exploiting an island of information. Average Entropy takes broader sweeps of the area thus gaining more information about the world. QVALAGG shows a non-trivial behavior exploiting one island before moving to another.

The same trend can be observed in Problem HIDDENCON. Figure 8(d) shows that the heuristic Average Entropy performs the best, while QVALAGG is able to match the heuristic. Rear Side Voxel saturates early on and performs worse. Figure 8(f) shows a similar trend as Figure 7(f).

6.6. Case study B: Train on synthetic, test on real

To show the practical impact of our framework, we show a scenario where a policy is trained on synthetic data and tested on a real dataset. Figure 9(a) shows some sample worlds created in Gazebo to represent an office desk environment on which QVALAGG is trained. Figure 9(b) shows a dataset of an office desk collected by TUM Computer Vision Group (Sturm et al., 2012). The dataset is parsed to create a pair of pose and registered point cloud that can then be used to evaluate different algorithms. Figure 9(c) shows that QVALAGG outperforms all heuristics. Figure 9(f) shows how QVALAGG learns a desk exploring policy by circumnavigating around the desk. This shows the powerful generalization capabilities of the approach. In

Table 2. Results for Problems HIDDEN-UNC and HIDDEN-CON on a spectrum of 2D and 3D exploration problems. The train size is 100 and test size is 10. Numbers are the confidence bounds (for 95% CI) of cumulative reward at the final timestep. Algorithm with the highest median performance is emphasized in bold.

Dataset	Sample World Maps	Problem	RewardAgg / QvalAgg	Average Entropy	Occlusion Aware	Unobserved Voxels	Rear Side Voxels	Rear Side Entropy
Concentrated Parallel Lines (2D)		HIDDEN-UNC	**(0.42,0.45)**	(0.20, 0.22)	(0.06, 0.09)	(0.20, 0.25)	(0.36, 0.41)	(0.30, 0.34)
		HIDDEN-CON	**(0.18, 0.27)**	(0.16, 0.20)	(0.07, 0.09)	(0.14, 0.18)	(0.19, 0.24)	**(0.21, 0.26)**
Distributed Blocks (2D)		HIDDEN-UNC	**(0.37,0.41)**	(0.26, 0.30)	(0.11, 0.16)	(0.22, 0.29)	(0.22, 0.29)	(0.24, 0.28)
		HIDDEN-CON	(0.20, 0.26)	**(0.21, 0.26)**	(0.11, 0.16)	(0.15, 0.20)	(0.15, 0.18)	(0.16, 0.19)
Poisson Forest of Circular Discs (2D)		HIDDEN-UNC	(0.58, 0.61)	**(0.59, 0.62)**	(0.49, 0.53)	(0.39, 0.45)	(0.53, 0.55)	(0.42, 0.47)
		HIDDEN-CON	**(0.54, 0.59)**	**(0.54, 0.59)**	(0.42, 0.46)	(0.34, 0.41)	(0.37, 0.43)	(0.39, 0.44)
Tabular World of Rectilinear Blocks (2D)		HIDDEN-UNC	**(0.43, 0.53)**	(0.31, 0.35)	(0.20, 0.26)	(0.28, 0.35)	(0.35, 0.44)	(0.25, 0.31)
		HIDDEN-CON	**(0.27, 0.33)**	(0.26, 0.29)	(0.18, 0.23)	(0.21, 0.28)	(0.18, 0.24)	(0.21, 0.27)
Bookshelves and Tables (3D)		HIDDEN-UNC	**(0.14, 0.31)**	(0.01, 0.04)	(0.01, 0.04)	(0.01, 0.04)	(0.01, 0.22)	(0.01, 0.19)
		HIDDEN-CON	**(0.05, 0.24)**	(0.01, 0.04)	(0.01, 0.04)	(0.01, 0.04)	(0.01, 0.22)	(0.01, 0.19)
Cluttered Construction Site (3D)		HIDDEN-UNC	**(0.14, 0.20)**	(0.01, 0.12)	(0.01, 0.09)	(0.01, 0.09)	(0.01, 0.11)	(0.01, 0.10)
		HIDDEN-CON	**(0.08, 0.12)**	(0.01, 0.12)	(0.01, 0.09)	(0.01, 0.09)	(0.01, 0.11)	(0.01, 0.10)
Office Desk and Chairs (3D)		HIDDEN-UNC	**(0.69, 0.80)**	(0.46, 0.59)	(0.51, 0.63)	(0.51, 0.63)	(0.59, 0.67)	(0.61, 0.72)
		HIDDEN-CON	**(0.55, 0.72)**	(0.46, 0.59)	(0.48, 0.63)	(0.48, 0.63)	(0.43, 0.52)	(0.41, 0.53)

contrast, the best heuristic Occlusion Aware gets stuck in a local minimum (Figure 9(e))

6.7. *Case study C: Policy search versus imitation learning*

We compared our approach with a baseline approach of policy search. We picked the problem setting HIDDEN-UNC, the dataset "Concentrated Parallel Lines" and the trained policy using REWARDAGG. We created a parametrized policy that was linear on the space of the information gain heuristics. The policy, parameterized by $\theta \in \mathbb{R}^6$, assigns at time t to each vertex v, picks the action with the highest score as follows

$$\arg\max_{v_t \in \mathcal{V}} \theta^T \mathcal{IG}(v_t)$$

We train such a policy using a black-box sample efficient policy search method, the covariance matrix adaptation evolution strategy (CMAES) Hansen (2016). CMAES is allowed 1,000 roll-outs, the same number of calls to oracle as REWARDAGG (note that CMAES actually has access to more information as they are full roll-outs compared with single reward calls in REWARDAGG). Figure 10(a) shows the comparison between the final policy trained by CMAES and the best policy on validation trained by REWARDAGG on a held out test dataset. We see that REWARDAGG outperforms CMAES not only on the cumulative reward by also at each timestep. This confirms our hypothesis that model-free policy improvement is slow to converge on account of sample inefficiency. It should be noted that the CMAES policy outperforms all the baseline heuristics as expected.

6.8. *Case study D:* FORWARDTRAINING *versus* AGGREVATE

We compared the training framework of FORWARDTRAINING, which trains a different policy for every timestep with AGGREVATE that trains a single policy across all timesteps. We wished to examine the following question: "How much data does a single AGGREVATE policy need to be competitive with FORWARDTRAINING?" We picked the problem setting HIDDEN-UNC and the dataset "Concentrated Parallel Lines." We trained FORWARDTRAINING where each policy π_t is given 500 datapoints (hence, for episode length $T = 30$, a total of 15,000 datapoints are used). We trained REWARDAGG where each iteration has 100 datapoints, and the number of iterations is 10. Hence, the REWARDAGG policy matches the same datasize as FORWARDTRAINING at iteration 5. Figure 10(b) shows a comparison between FORWARDTRAINING and REWARDAGG. We see that REWARDAGG outperforms FORWARDTRAINING by iteration 4, following which the performance converges and oscillates at values above FORWARDTRAINING. Interestingly, at iteration 5 REWARDAGG outperforms FORWARDTRAINING even though each policy in FORWARDTRAINING has access to the same dataset size as REWARDAGG.

We conjecture that this might be because of the generalization effect across timesteps: FORWARDTRAINING might be over-fitting as it reasons about timesteps individually.

6.9. Case study E: Homogeneous versus heterogeneous datasets

We examined the behavior of policies trained on homogeneous versus heterogeneous datasets. We wished to examine the following question: "Can we learn a policy that is able to distinguish between the two type of environments after some initial measurements?" We picked two distinct datasets: D1 (parallel lines) and D2 (distributed blocks). Policy-D1 is trained only on D1 and Policy-D2 is trained only on D2. We train a third policy, Policy-D1+D2 by combining both datasets (but reducing the size by half). Figure 11 shows a comparison between the policies on both datasets. As expected, Policy-D1 and Policy-D2 perform the best on a test problem from their respective datasets. However, Policy-D1+D2 is competitive with these policies and outperforms baseline heuristics. On examining the snapshots, we see that after the first few timesteps, Policy-D1+D2 is able to distinguish between the two environments and adapt accordingly. Figure 11(c) shows the policy focusing on the parallel lines in the center after it discovers it. Figure 11(d) shows the policy examining the periphery of the world map.

7. Experiments on search-based planning

In this section, we evaluate our approach extensively on a set of search-based planning problems for 2D planning on synthetic problems and more realistic 4D non-holonomic path-planning problems encountered by UAVs flying at various speed regimes. We choose a wide variety of world distributions ranging from simple and intuitive environments, chosen to highlight the importance of exploiting environment structure in motion planning, to complex, heterogenous environments for analyzing scalability and robustness. We also present closed-loop results on a UAV flying outdoors at high speeds.

In addition, we have developed a simple and intuitive Python-based planning pipeline to serve as a backend for the Gym environment. The planning environment exposes search as a policy and makes it easy to incorporate standard machine learning libraries (Abadi et al., 2016; Theano Development Team, 2016) with custom planning graphs that require only environment images as input. We use this planning pipeline to conduct all our experiments. Source code and instructions can be found via our project page at https://goo.gl/YXkQAC

7.1. Problem details

We first describe the 2D navigation task. Here, the world map ϕ is a 2D binary map. The graph $\mathcal{G}$ is a discrete lattice of size 200×200 where each node is connected to the 8 neighbors. The robot has to plan from bottom-left to top-right of the lattice. Note that while the grid size for these problems are small, the edge evaluation for such a graph could be arbitrarily expensive in practice. For example, consider the problem of planning 2D routes for aircrafts. It is plausible to envision that the lattice resolution is 100 m and the 200×200 lattice covers an area of 20 km. Evaluation of each edge of such a lattice requires collision checking with other dynamically moving aircraft, no-fly zones, and risk of flying over urban areas. This implies that a real-time traffic control can only search a small fraction of the lattice.

We now describe a more realistic 4D non-holonomic path-planning problem on a state lattice for problems encountered by UAVs. The term *non-holonomic path planning* (Laumond et al., 1998) refers to the fact that certain class of dynamical systems are constrained in the range of feasible motions the robot can execute (Kelly and Nagy, 2003). It is a common practice to approximate UAVs moving at high speeds as curvature constrained systems with unicycle dynamics (Choudhury et al., 2014; Dugar et al., 2017a,b). We consider the problem of path planning for such systems by planning on a state lattice (Pivtoraiko et al., 2009). We consider two classes of UAVs: an autonomous helicopter moving at speeds of 30 m s^{-1} and a quadrotor (DJI M100) flying at 5 m s^{-1}.

The autonomous helicopter has a minimum radius of 50 m and plans on a state lattice $\mathcal{G}$ of resolution 25 m. The average degree of a node is 21. The distance between start and goal is 600 m. The world ϕ is represented as a 3D binary grid map and a set of 3D no-fly zones (represented as polygons with a height range). An edge evaluation requires that every state on an edge is at a clearance distance from all obstacles. Expansion of each node takes $\sim$1 ms on average. The robot is required to plan within a time budget of 500 ms, thus corresponding to a maximum of 500 expansions.

The quadrotor has a minimum radius of 12.5 m and plans on a state lattice $\mathcal{G}$ of resolution 12.5 m. The average degree of a node is 9. The distance between start and goal is 300 m. The world is represented as a 3D binary grid map and a set of 3D no-fly zones. Expansion of each node takes $\sim$1 ms on average. The robot is required to plan within a time budget of 1,000 ms, thus corresponding to a maximum of 1,000 expansions.

7.2. Baseline approaches for search-based planning

7.2.1. Motion planning baselines. For 2D navigation, we compare against greedy best-first search with two commonly used heuristics: the Euclidean distance (h_{EUC}) and the Manhattan distance (h_{MAN}). We also use the A* algorithm as a baseline with h_{EUC} heuristic. In addition, we compare against the MHA* algorithm (Aine et al., 2016), which has been proven to be an effective way of combining multiple, often unrelated, heuristics providing bounds

on solution quality (Phillips et al., 2015). We use a simplified version that 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 $\mathcal{I}$.

For 4D non-holonomic planning problems, we use the Dubins distance (Dubins, 1957) as a heuristic.

7.2.2. Machine learning baselines. We consider two learning baselines (a) supervised learning (SL) with data from roll-outs with π_{OR} and (b) RL using evolutionary strategies (CEM) and Q-Learning (QL) with function approximation. These methods are explained in detail in Appendix 8.4.

7.3. Imitation learning details

7.3.1. Feature extraction and learner. The policy maps the history ψ to an action a by learning a function approximation for the action value function $\hat{Q}(\psi, a)$. The tuple (a, ψ) is mapped to a vector of features f. Here the history ψ is represented as a concatenation of all lists, i.e. $\psi_t = \{\mathcal{O}, \mathcal{C}, \mathcal{I}\}$. The action a is the vertex v to expand.

We now describe the feature extraction for 2D navigation problems. Although, technically, the features for a vertex v should depend on the parent edge e that leads to the vertex, we ignore this in practice and consider a vertex in isolation to calculate features. It is important to note that the features used must be easy to calculate (no high computational burden) and should only require information uncovered by search until that point in time (otherwise it would count as extra expansions). We define the feature vector to be a concatenation of the two vectors, i.e. $f = [f_S, f_E]$.

Search-based features: $f_S(v, \psi)$. These features depend on the state of the search only and do not probe the environment.

- (x_v, y_v): location of a node in the coordinate axis of the occupancy map.
- (x_{v_g}, y_{v_g}): location of the goal in the coordinate axis of the occupancy map.
- g_v: cost (number of expansions) of the shortest path to start.
- h_{EUC}: Euclidean distance to the goal.
- h_{MAN}: Manhattan distance to the goal.
- d_{TREE}: depth of v in the search tree so far.

Environment-based features: $f_E(v, s)$. These features depend upon the environment uncovered so far, more specifically the vertices in $\mathcal{I}$.

- $x_{OBS}, y_{OBS}, d_{OBSX}$: coordinates and distance of the closest node in $\mathcal{I}$ to v.
- $x_{OBSX}, y_{OBSX}, d_{OBSX}$: coordinates and distance of the closest node in $\mathcal{I}$ to v in terms of x-coordinate.
- $x_{OBSY}, y_{OBSY}, d_{OBSY}$: coordinates and distance of the closest node in $\mathcal{I}$ to v in terms of y-coordinate.

We discuss more about alternate representations and feature extraction ideas in Appendix 8.4.

For the 4D planning problems, we use a slightly altered feature representation that is eight-dimensional.

- Normalized Euclidean distance to the start.
- Normalized Euclidean distance to the goal.
- Dot product between start, vertex, and goal.
- Normalized Dubins distance to the start.
- Normalized Dubins distance to the goal.
- Normalized heading of a vertex.
- Normalized distance of a vertex from the closest obstacle.
- Dot product between distance to an obstacle and heading of a vertex.

Such a feature representation is chosen as these terms are easy to compute and are informative in estimating the utility of expanding a vertex.

The learner is represented using a feed-forward neural network with two fully connected hidden layers containing $[100, 50]$ units and ReLu activation. The model takes as input a feature vector $f \in \mathcal{F}$ for the pair (v, s) and outputs a scalar cost-to-go estimate. The network is optimized using RMSProp Tielman and Hinton (2012). A mini-batch size of 64 and a base learning rate of 0.01 is used. The network architecture and hyper-parameters are kept constant across all environments. For experiments with the UAV, we use a random forest regression (Liaw and Wiener, 2002).

7.3.2. Dataset creation. The 2D world maps are created by randomly distributing geometric objects such as rectangles and circles according to a hand-designed parametric distribution. Each environment class is representative of challenging artifacts in motion planning such as narrow corridors, local minimum, single homotopies. Heterogeneous environments are created to show that the heuristic can deal with such problems as well.

For the experiment with a real robot, a dataset of mazes was created and a real-life maze was simulated using no-fly zones.

7.3.3. Clairvoyant oracle. We use the backward Djikstra algorithm as the clairvoyant oracle. It is executed until it expands to all states, or until it reaches a cost-to-go limit. We can further reduce these states by using a backwards weighted A* instead of Dijkstra. Although this might induce suboptimality in expansion effort, it would still be useful in domains where we only require the cost-to-go for a subset of the space. For every state not reached, we can set the label as a maximum expansion-to-go. We note that using such an oracle for higher dimensions might be infeasible in higher dimensions and discuss remedies in Section 8.

7.3.4. Practical algorithm implementation. As the size of the action space changes as more states are expanded,

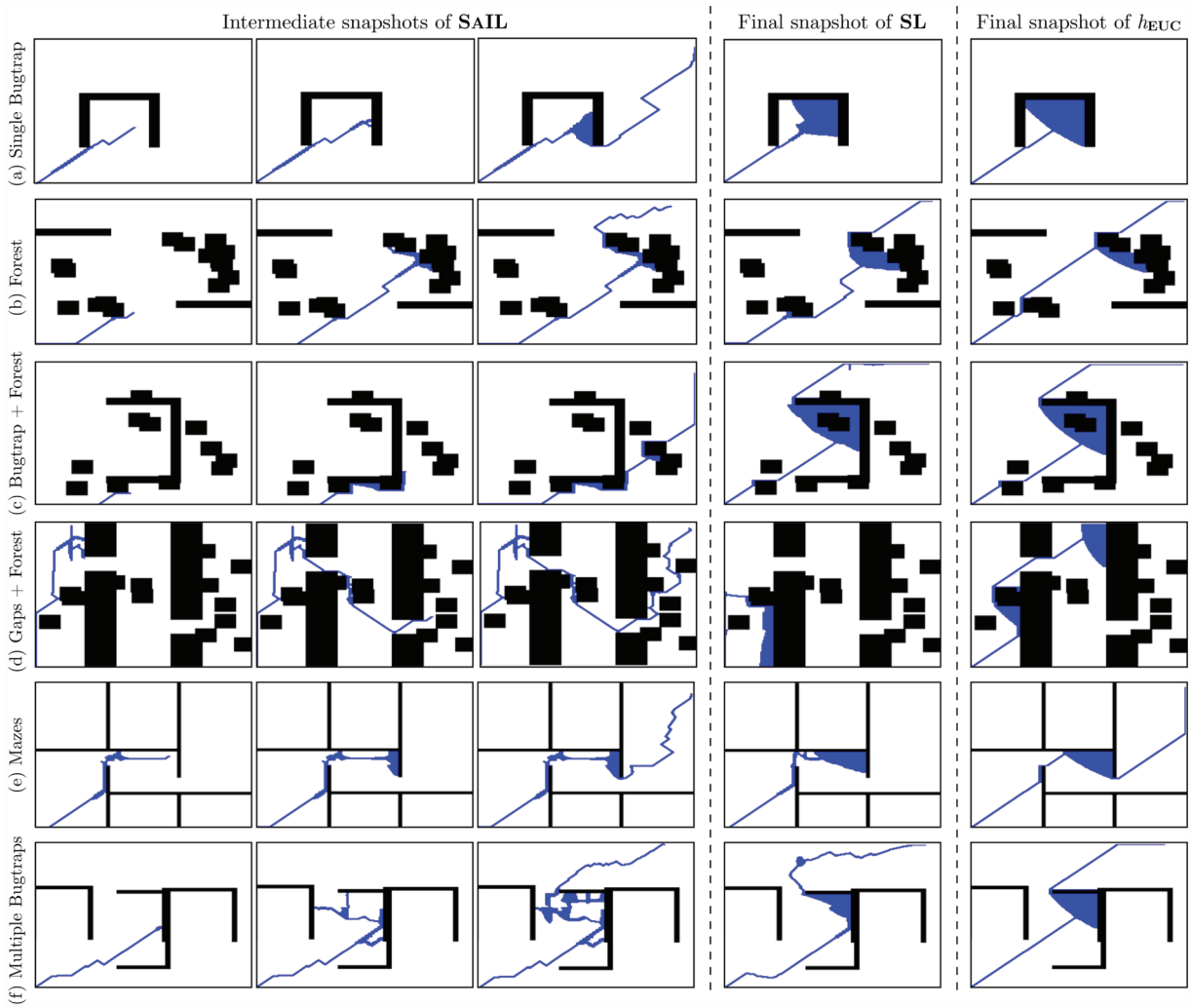


Fig. 12. Evolution of search frontier (expanded (blue), invalid (black), unexpanded (white)) of SAIL compared with final snapshot of supervised learning (SL) and h_{EUC} across all environments. SAIL expands far fewer states.

the SAIL algorithm requires a forward pass through the model for every action individually unlike the usual practice of using a network that outputs cost-to-go estimate for all actions in one pass as in Mnih et al. (2015). This can become computationally demanding as the search progresses ($\mathcal{O}(N)$ in actions). Instead, we use a *priority queue* as $\mathcal{O}$, which sorts vertices in increasing order of the cost-to-go estimates as is usually done in search-based motion planning. The vertex on the top of the list is then expanded. We use two priority queues, sorted by the learner and oracle's cost-to-go estimates, respectively. This allows us to take actions in $\mathcal{O}(1)$ but forces us to freeze the Q-value for a vertex to whenever it is inserted in $\mathcal{O}$. Despite this artificial restriction over the policy class Π, we are able to learn efficient policies in practice. However, we wish to relax this requirement in future work. We also analyze the time complexity in Appendix 8.4.

7.4. Analysis of results

Table 3 shows the normalized evaluation cost of all algorithms on various datasets. Snapshots of planning with different heuristics are shown in Figures 12 and 13(a). Convergence of different learning algorithms are shown in Figure 13(b). We present a set of key observations to summarize these results.

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

Table 3 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.

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

Table 3. 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, 5,000] (maximum possible: 40,000). Planning parameters are: map size 200 × 200, T_{train} = 1,100, T_{test} = 20,000. Data sizes are: training (200), test (100), validation (70). NONAMEparameters are: k 50, $\beta_0 = 0.7$. NONAME, CEM, and QL are run for $N = 15$ iterations. SL uses $m = 600$.

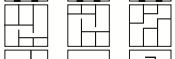
Dataset	Sample Worlds	SAIL	SL	CEM	QL	h_{EUC}	h_{MAN}	A*	MHA*
Alternating Gaps		**0.039**	0.432	0.042	1.000	1.000	1.000	1.000	1.000
Single Bugtrap		0.158	0.214	**0.057**	1.000	0.184	0.192	1.000	0.286
Shifting Gaps		**0.104**	0.464	1.000	1.000	0.506	0.589	1.000	0.804
Forest		**0.036**	0.043	0.048	0.121	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 13(b) shows that on the "Forest" dataset, SAIL converges by the sixth iteration, whereas CEM takes 12 iterations and QL does not converge. SAIL also converges quickly (by the eighth iteration) across datasets.

Observation 3. SAIL *is able to detect and escape local minima.*

A classic case in motion planning is the bugtrap (Figure 1(b)) that traps a greedy search in a local minimum. Figure 12(a) and (f) shows that when trained on such distributions, SAIL is able to detect these artifacts and smartly escape them by exploring in different directions.

Observation 4. SAIL *is able to exploit the relative configuration of obstacles and environment structure.*

In a maze world with rectilinear hallways (Figure 12(e)), SAIL learns to quickly find a wall and then concentrate the search along the axes. In Figure 12(d), SAIL focuses only on regions where there is a high probability of a gap and skids along obstacles otherwise.

7.5. Case study A: Adaptive behavior of SAIL

We take a closer look at the behavior of SAIL in response to a change in the distribution of worlds that it is being trained on $P(\phi)$. Consider the scenario illustrated in Figure 13(a). We create two datasets. Both datasets have a wall in the middle of the environment, with a gap in the wall. For dataset 1, the gap can occur uniformly randomly along the y-axis. For dataset 2, the gap either occurs with 70% probability at the bottom and 30% probability at the top.

For dataset 1, SAIL learns to approach the center of the environment first and then search along the wall until it finds a gap. This is in response to the fact that the gap can occur anywhere and, hence, this is a cost-efficient strategy. Contrast this to a greedy search that gets stuck expanding states near the top of the wall.

For dataset 2, SAIL learns to approach the bottom of the environment first and then search along the wall. This is in response to the gaps occurring at the bottom of the wall. The greedy search is non-responsive to the change in distribution and gets stuck expanding states near the top again.

7.6. Case study B: Helicopter path planning

An important application of heuristic learning is to speed up high-dimensional search. An application of particular relevance to us is an autonomous helicopter (Choudhury et al., 2014). A class of environment in which the helicopter has to plan in is a canyon-like environment. As the system moves at a speed of 30 m s^{-1}, it has to produce a plan in real time (within 200 ms) otherwise it risks reaching states from which collision is inevitable.

We use SAIL to learn a heuristic that guides search in such environments. We collect a dataset by generating canyons using a parametric distribution as showing in Figure 14(a). A lattice with the specifications described in Section 7.1 is created. As a baseline, we run A* with Dubins distance as the heuristic on this problem. As shown in Figure 14(b), this ends up expanding a large number of vertices (2,531). This is because the Dubins distance is not the optimal cost to do. The under-estimation of this heuristic results in a large number of vertices being expanded and, hence, a long planning time (7,000 ms).

We also run a greedy search using the Dubins distance as a heuristic. We see that for these kind of environments, greedy search performs pretty well: the number of vertices expanded is 142 and planning time is 500 ms. However, the greedy search expends search effort trying to search for a tunnel through the canyon.

SAIL has much better performance than either of these baselines. It is able to learn a heuristic that expands only 18 vertices with a search time of 100 ms. The features used

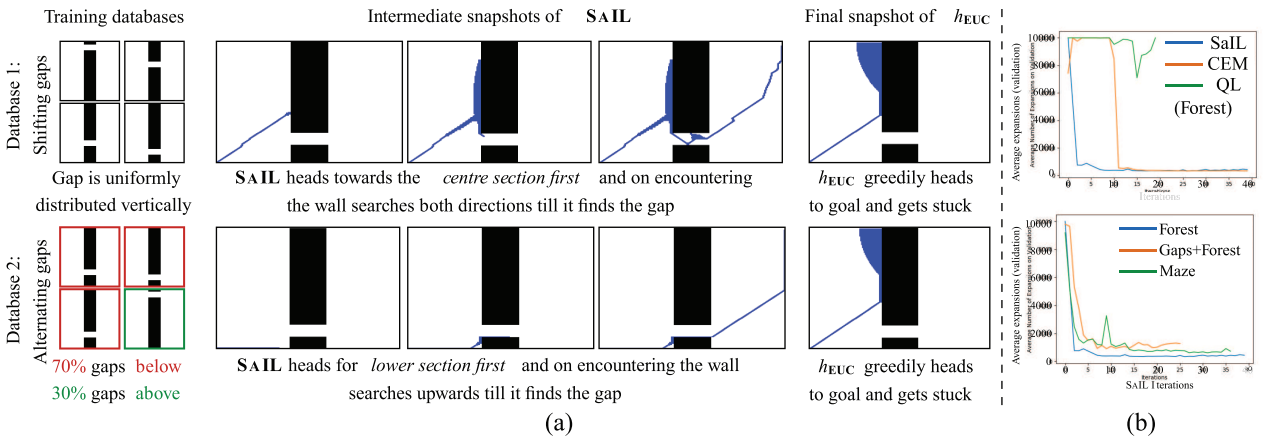


Fig. 13. (a) SAIL learns to adapt to different environment distributions by directing search to areas where it expects to find gaps. Note SAIL does not have information about the entire environment, only the explored part. (b) On the "Forest" dataset, SAIL converges faster that CEM and QL to a good policy. SAIL also converges consistently to a good policy across environments "Gaps," "Gaps+Forest," and "Maze."

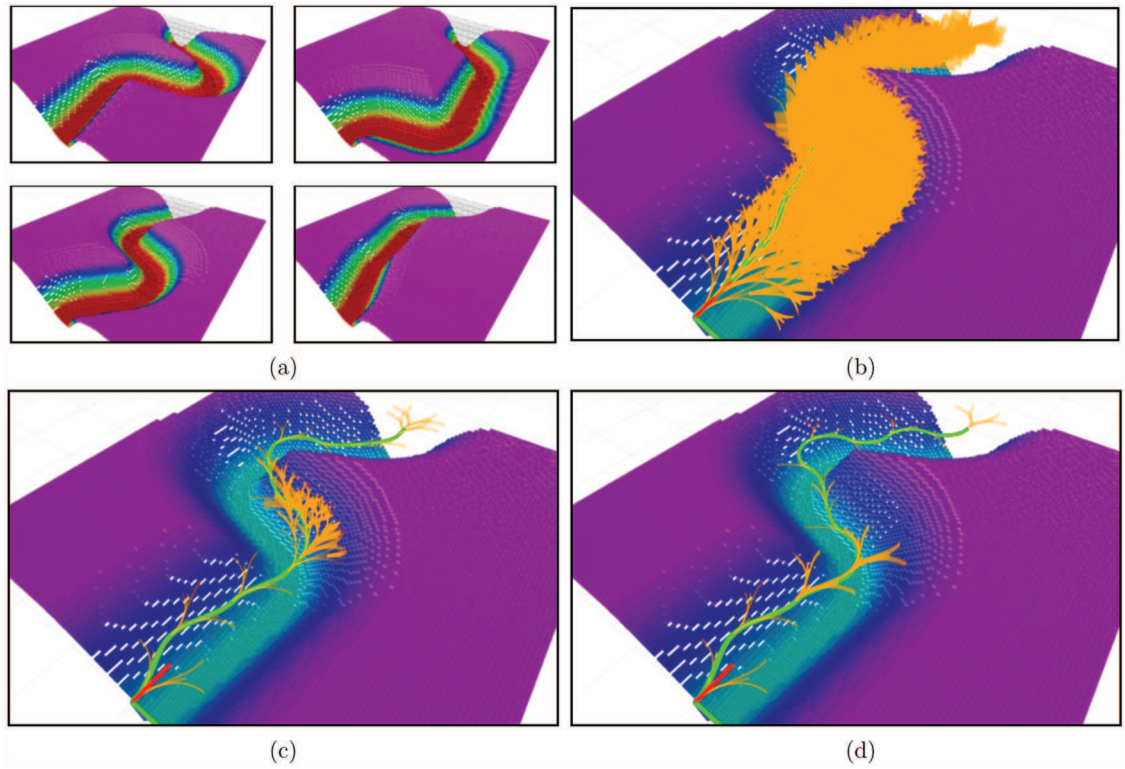


Fig. 14. Experiments on path planning for an autonomous helicopter in a canyon environment. The environment is motivated from planning challenges as described in Choudhury et al. (2014). (a) Dataset of canyon-like environments generated by a parametric distribution. (b) The search tree from A* with Dubins distance heuristic on a test environment. The start point is shown by the axes. The expanded edges are shown in yellow. The planned path is shown in green. A* expands 2,531 vertices and takes 7,000 ms. (c) The search tree for greedy search with Dubins distance heuristic. It expands 142 vertices and takes 500 ms. Note that most of the wasted expansions are where the tree tries to search through the canyon wall (d) SAIL expands only 18 vertices and takes 100 ms. It hugs the canyon wall until it reaches the goal.

by SAIL are minimalistic and are enlisted in Section 7.3.1. Among those features are the Dubins distance to the goal and the direction vector to the nearest obstacle. By examining the search tree produced by SAIL, we observe that it learns a trade-off between following the Dubins distance heuristic and not expanding states that are pointing into the canyon wall (as such states would not result in a feasible path eventually).

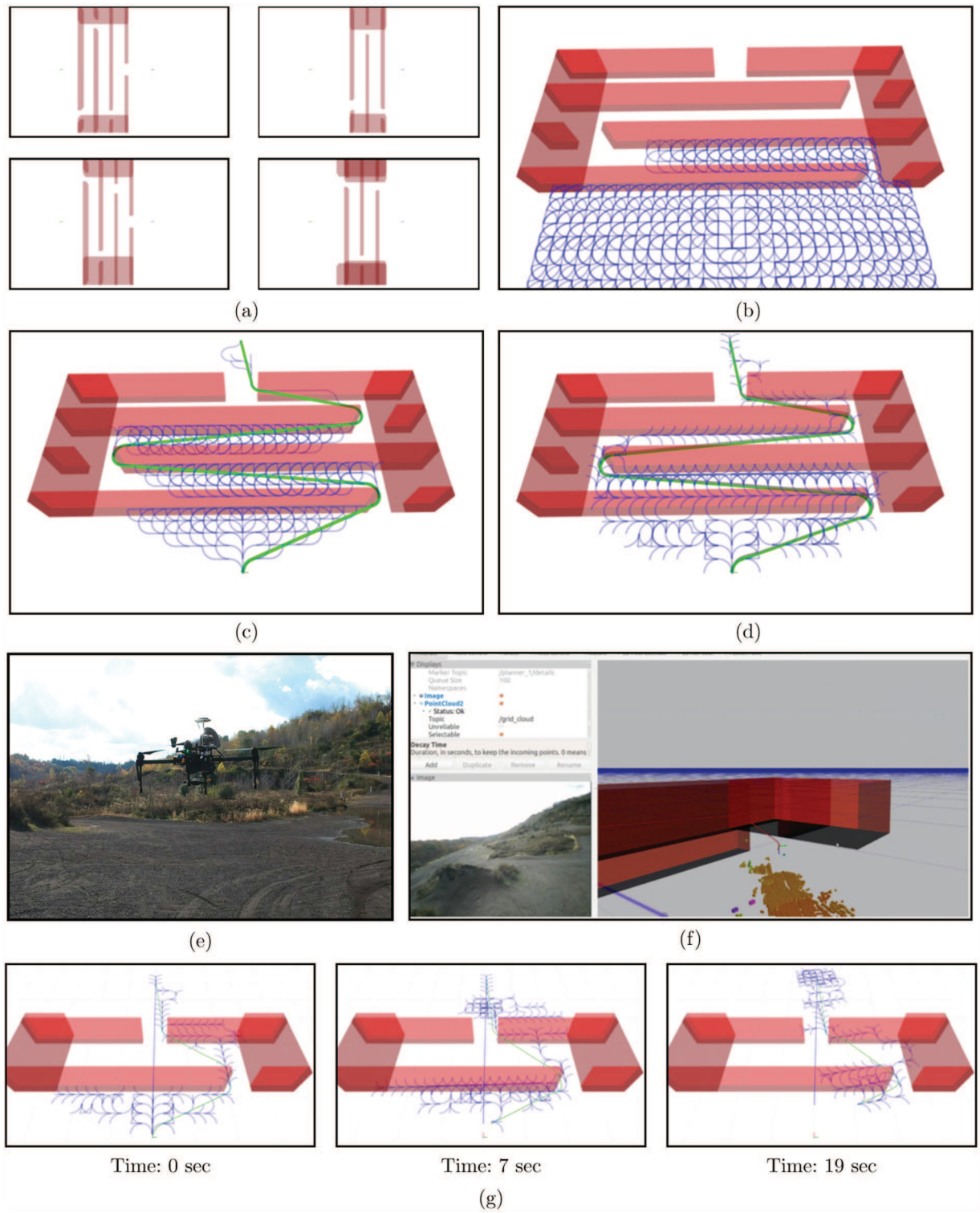


Fig. 15. Experiments on path planning for a real quadrotor flying at high speed 5 m s^{-1} while avoiding no fly zones that represent a maze-like scenario. (a) A dataset of mazes created from a parametric distribution. (b) The search graph of A* on the environment. It expands 1,910 states in the 1,000 ms time budget without finding a path. (c) The greedy search with Dubins distance expands 661 vertices and takes 400 ms. The remaining time is used to relax the path shown in green. (d) SAIL outperforms both and finds a path by expanding only 180 vertices in 120 ms. (e) The DJI M100 used for our experiments. (f) An experiment where SAIL is running onboard the robot. A set of no-fly zones is created and the robot has to fly through it. The robot view and onboard imagery is shown. (g) A time lapse of the search tree as the robot replans while performing the mission. We can see that the search tree remains sparse through out and SAIL is always able to find a path.

7.7. Case study C: Quadrotor path planning

We also applied this approach to a real quadrotor that has to navigate in an environment at high speed 5 m s^{-1} while avoiding no-fly zones. No-fly zones can result from areas that a UAV cannot fly over because of risks to property or from other vehicles in the area. These no-fly zones can be arbitrary in complexity thus creating artifacts such as a maze as shown in Figure 15. The motivation for including these results is to show the utility of this approach on real-world planning problems where computational resources are limited. We also highlight that since we are learning a policy for planning, the simulation to real-world transfer is made far easier as the learning only has to generalize across geometry of obstacles (as opposed to policies that map sensor measurements to actions).

We create a dataset of such mazes by means of a parametric distribution as shown in Figure 15(a). We give a time budget of 1,000 ms for planners to solve the problem. A* with Dubins heuristic is unable to solve the problem in the time limit as shown in Figure 15(b). This is because the Dubins distance vastly under-estimates the distance to the goal in this environment. A* expands 1,910 states before being terminated.

Greedy search with Dubins heuristic is able to find a path after 661 expansions within the time budget (in 400 ms). The remaining time is spent relaxing the path found. The greedy behavior is beneficial in this environment because it results in a wall-following-like behavior. However, the algorithm wastes search effort expanding states perpendicular to the wall that would lead to inevitable collision.

SAIL outperforms both algorithms by finding a path in 180 expansions (in 120 ms). The remaining time is spent relaxing the path. As can be seen for the search graph, it focuses on expanding paths perpendicular to the wall. It learns to not expand vertices that point into the wall since the oracle shows the cost to go of such nodes to be ∞.

We also evaluated SAIL on board a DJI M100 quadrotor equipped with a TX2 computer. We created a synthetic maze with no-fly zones and commanded the robot to fly through it (Figure 15(e) and (f)). SAIL is able to find a path expanding a sparse number of vertices. As the robot follows the path, the algorithm is able to consistently replan and find a path consistently without expanding too many states (Figure 15(g)).

8. Discussion and future work

We have presented a novel data-driven IL framework to learning planning policies. Our approach trains a policy to imitate a clairvoyant oracle that has full information about the world and can compute optimal planning decisions. We examined two problem domains: IPP and search-based planning. We evaluated our approach in both these domains and showed that the learnt policy can outperform state-of-the-art approaches. We now discuss a set of relevant questions and directions for future work.

8.1. When does this framework lead to good policies? What are some failure cases?

MDP framework provides an elegant way of posing problems where the complete state of the problem space is known. The value of an action for a given state in an MDP is given by

$$Q_t^\pi(s,a) = R(s,a) + \mathbb{E}_{s' \sim P(s'|s,a)}\left[V_{t-1}^\pi(s')\right] \tag{22}$$

$$V_t^\pi(s) = \sum_{i=t}^{T} \mathbb{E}_{s_t \sim P(s_i|\pi,i,s)}\left[R(s_i, \pi(s_i))\right] \tag{23}$$

The optimal MDP policy maximizes the expected cumulative reward, i.e. $\pi^*(s_t) \in \arg\max_{\pi \in \Pi} V_t^\pi(s_t)$.

However there are two major challenges that POMDP solvers face.

- Computing the expectation over the state space. As the state space of most of the problems worth solving is large, computing an expectation over such state space needs a high number, making it expensive to evaluate online.
- Keep track of evolving uncertainty about the state space over the planning horizon.

Our approach solves the first challenge through data-driven techniques: the MDP solvers used are over-sampled MDP problems to train a policy on the expected distribution of problems. The hallucinating oracle is similar in nature to a QMDP algorithm (Littman et al., 1995), an effective approximate solution to POMDPs, which takes the best action on the current posterior. However, although QMDP requires maintaining an explicit posterior, our framework does not. QMDP has been shown to be very successful where explicit information-gathering behavior is not required (Koval et al., 2014; Javdani et al., 2015): the belief collapses irrespective of the action. Hence, this optimization assumes a fixed belief and does not account for evolving belief over time (which is the second challenge for POMDPs). This implies there is no motivation for the MDP solver and, hence, the learnt policy to change the belief.

These kind of methods work quite well in POMDP problems where the required changes in belief can be attained by actions that are rewarding as well. 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. There exists no action that is not rewarding while reducing the uncertainty. We note that this is not true for all learning in planning paradigms. For example, when learning to collision check (Choudhury et al., 2017a), a policy that actively reduces uncertainty about the world is effective.

To illustrate the failure case, we present a simple scenario as shown in Figure 16. We have a "trapped robot" whose task is to escape from a room, i.e. it receives a reward for

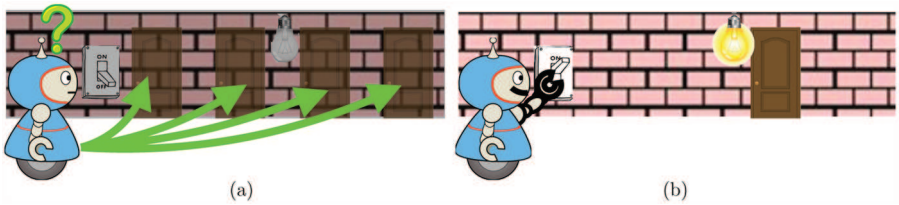


Fig. 16. The robot in a dark room problem. The robot is uncertain about the location of a door and the only way to collapse that uncertainty is to pull a light switch. (a) A clairvoyant oracle is not incentivized to flip the switch and, hence, the robot does not learn to collapse uncertainty. (b) The optimal POMDP policy would be to flip the switch and then head for the door.

escaping and penalization for staying in the room. The room is dark, i.e. the robot cannot observe the location of the door. It can perform actions such as moving in the room. It can also perform an action of flicking on the light switch. On performing such an action, it receives an observation containing the location of the door. An optimal POMDP policy would always choose this action, collapse uncertainty about the door location and subsequently head straight for the door. However, imitation of clairvoyant oracles do not provide such behaviors. The oracle, at training, always guides the robot towards the door to maximize reward and is not incentivized to flip the light switch. The policy learns a blind search pattern that takes a long time to find the door.

For such POMDP problems, one way forward would be to incentivize the oracle at train time to reduce the uncertainty as suggested by the POMDP-lite approach (Chen et al., 2016a). Although POMDP-lite quantifies uncertainty reduction as L1-norm of the belief change, this can be hard to compute for the space of world maps. Using approximations to this belief change would be an interesting direction of future work.

8.2. How can we incorporate solution cost in addition to search effort in this framework?

Although our framework ignores 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 (Kuffner and LaValle, 2000). 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*) (Aine et al., 2016) can be employed. In such a framework, any (uncalibrated or inconsistent) heuristic function (Narayanan et al., 2015) can be used in tandem with an anchored search that uses an inflated admissible heuristic. This ensures that the found path is bounded suboptimal. We can follow the same strategy of mapping such problems to a POMDP where we penalize each expansion until the anchor search pops the goal vertex. The transition function is more complicated as it involves the mechanics of MHA*. However, it would be interesting to see the nature of the learnt policy in this setting.

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* (Choudhury et al., 2016): finds a feasible solution quickly and iteratively improves it. In this paradigm, SAIL trains a heuristic that displays a behavior 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.

8.3. Can we generalize this framework to sampling-based

The SAIL framework defines `Search` in a very general way: the underlying implicit graph can also be a tree and the expansion operation can be a local steering operation akin to the framework of EST (Hsu et al., 1999). The oracle design is an open question: a plausible oracle is growing a backward tree from the goal and using a k-nearest-neighbor (k-NN) value function approximator. Another paradigm to consider is when the `Expand` operation is a call to a *sampler*. For example, the framework in randomized A* (RA*) (Diankov and Kuffner, 2007) proceeds by selecting a node of the search tree using some criteria and sampling around it.

Recently, Ichter et al. (2017) proposed a framework for learning sampling distributions from optimal paths during training by using a conditional variational auto-encoder (CVAE). However, in this framework sampling and planning are decoupled, i.e. the sampling policy learns a good stationary distribution from which samples are generated and provided to the planner. Hence, the planner does not adapt during the planning cycle. Such a stationary distribution can be very hard to learn as directly predicting the optimal path requires conditioning on a lot of information about the environment.

SAIL can be extended to learn sampling policies that address this problem. The CVAE can condition on the state of the search (similar to the feature vector used by SAIL). The labels can be obtained by a backward tree from the goal grown during training. The iterative learning process

of SAIL will ensure that the CVAE is trained on the distribution of search state actually encountered rather than simply using the optimal path.

8.4. Incorporating noise in transition and observation for IPP problems

The IPP problem that we defined in Section 2.1.1 and subsequently mapped to a POMDP in Section 3.2 consider a deterministic measurement and utility function. This can always be relaxed in an ad-hoc way: the occupancy map used to represent ψ_t is essentially a Bayes' filter and can handle noisy observations, and the policy can also handle motion uncertainty since during the training phase, data collected in the initial stages is from random motions of the learner.

However, if one is to formally incorporate noise, the mapping needs to be re-examined. The crucial change arises from the fact that the utility function $\mathcal{F}$ is no longer dependent only on the sequence of vertices visited $\{v_i\}_{i=1}^t$ and the world ϕ. It also depends on the actual observations received $\{y_i\}_{i=1}^t$, i.e. the utility function needs to be redefined to have the following arguments $\mathcal{F}\left(\{v_i, y_i\}_{i=1}^t, \phi\right)$.

To provide a concrete example, we re-examine our application of 3D reconstruction of objects in the environment presented in Section 6.1. We had assumed that each vertex v_i in a path ξ is associated with a unique measurement y_i. The union of all measurements defined the coverage map C that, in turn, defined the utility. As this unique measurement assumption is no longer true, the coverage map has to explicitly consider the actual measurements received. This results in a utility function $\mathcal{F}\left(\{v_i, y_i\}_{i=1}^t, \phi\right)$ that depends on measurements as well.

Keeping this important change in mind, we redefine the mapping to POMDP. The state is defined to contain all information that is required to define the reward, observation, and transition functions. Let the state be the set of nodes visited and *measurements received* as well as the underlying world, $s_t = \{\{v_i, y_i\}_{i=1}^t, \phi\}$. At the start of an episode, a world is sampled from a prior distribution $\phi \sim P(\phi)$ along with a graph $\mathcal{G} \sim P(\mathcal{G})$. The initial state is assigned by setting $s_1 = \{v_1, y_1, \phi\}$. Note that the state s_t is partially observable due to the hidden world map ϕ.

We define the action a_t to be the next *desired node* to visit. The reward function is now defined as a function of the state s_t only as the marginal gain in utility on receiving $\{v_t, y_t\}$. The marginal gain of the utility function $\mathcal{F}$ is $\Delta_{\mathcal{F}}\left(\{v_t, y_t\} \mid \{v_t, y_t\}_{i=1}^{t-1}, \phi\right) = \mathcal{F}\left(\{v_i, y_i\}_{i=1}^t, \phi\right) - \mathcal{F}\left(\{v_i, y_i\}_{i=1}^{t-1}, \phi\right)$. In addition, the reward is set to $-\infty$ whenever the cost budget is violated, i.e.

$$R(s,a) = \begin{cases} \Delta_{\mathcal{F}}\left(\{v_t, y_t\} \mid \{v_t, y_t\}_{i=1}^{t-1}, \phi\right) & \text{if } \mathcal{T}\left(\{v_i\}_{i=1}^t, \phi\right) \leq B \\ -\infty & \text{otherwise} \end{cases} \tag{24}$$

The state transition function, $\Omega\left(s, a, s'\right)$, is defined by the execution model $P(v_{t+1} | a_t, \phi)$ and the measurement model $P(y_{t+1} | v_{t+1}, \phi)$. Given state s_t, the observation is now deterministic because it is contained in the state, i.e. $o_t = y_t$.

There are several challenges with this formulation. Take the utility function $\mathcal{F}\left(\{v_i, y_i\}_{i=1}^t, \phi\right)$ for instance. This might be an invocation of an optimization subroutine to reconstruct the 3D structure that is then compared against the ground truth ϕ. Such a function is not likely to possess the submodularity property as before. This ends up affecting the choice of the clairvoyant oracle: the GCB algorithm used previously relied on the submodularity property for its effectiveness. It is likely that a suitable approximation must be applied for such problems, and this topic needs further analysis.

Acknowledgment

We would like to thank Silvio Maeta, Vishal Dugar, and Brian McAllister for help with flight test results on the UAV. We would like to thank Shushman Choudhury for insightful discussions and feedback on the paper.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: We would like to acknowledge the support from the ONR (grant number N000141310821) and NASA (contract number NNX17CS56C).

Notes

1. Of course, the direct solutions from such planners are usually not usable and require post processing.
2. See https://bitbucket.org/sanjiban/matlab_learning_info_gain
3. Assuming that a computationally inexpensive local planner exists to compute such an arc.

References

Abadi M, Agarwal A, Barham P, et al. (2016) Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR* abs/1603.04467.

Abbeel P and Ng AY (2004) Apprenticeship learning via inverse reinforcement learning. In: *Proceedings of the 21st International Conference on Machine Learning*. New York: ACM Press.

Aine S, Swaminathan S, Narayanan V, Hwang V and Likhachev M (2016) Multi-heuristic A. *The International Journal of Robotics Research* 35: 224–243.

Arfaee SJ, Zilles S and Holte RC (2011) Learning heuristic functions for large state spaces. *Artificial Intelligence* 175: 2075–2098.

Arora S and Scherer S (2017) Randomized algorithm for informative path planning with budget constraints. In: *Proceedings of ICRA*.

Arulkumaran K, Deisenroth MP, Brundage M and Bharath AA (2017) A brief survey of deep reinforcement learning. *Preprint arXiv:1708.05866*.

Asmuth J and Littman ML (2011) Approaching Bayes-optimalilty using Monte-Carlo tree search. In: *Proceedings 21st International Conference on Automated Planning and Scheduling*, Freiburg, Germany.

Bhardwaj M, Choudhury S and Scherer S (2017) Learning heuristic search via imitation. In: *Proceedings of CoRL*.

Boots B, Siddiqi SM and Gordon GJ (2011) Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research* 30(7): 954–966.

Brafman RI and Tennenholtz M (2002) R-max—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3: 213–231.

Canny J (1988) *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press.

Chang KW, Krishnamurthy A, Agarwal A, Daume H and Langford J (2015) Learning to search better than your teacher. In: *Proceedings of ICML*.

Charrow B, Kahn G, Patil S, et al. (2015) Information-theoretic planning with trajectory optimization for dense 3D mapping. In: *Proceedings of RSS*.

Chekuri C and Pal M (2005) A recursive greedy algorithm for walks in directed graphs. In: *Proceedings of FOCS*.

Chen M, Frazzoli E, Hsu D and Lee WS (2016a) POMDP-lite for robust robot planning under uncertainty. *Preprint arXiv:1602.04875*.

Chen Y, Hassani SH and Krause A (2016b) Near-optimal bayesian active learning with correlated and noisy tests. *CoRR* abs/1605.07334.

Chen Y, Javdani S, Karbasi A, Bagnell J, Srinivasa S and Krause A (2015) Submodular surrogates for value of information. In: *Proceedings of AAAI*.

Choudhury S, Arora S and Scherer S (2014) The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety. In: *AHS 70th Annual Forum*, Montreal, Quebec, Canada.

Choudhury S, Dellin CM and Srinivasa SS (2016) Pareto-optimal search over configuration space beliefs for anytime motion planning. In: *Proceedings of IROS*.

Choudhury S, Javdani S, Srinivasa S and Scherer S (2017a) Near-optimal edge evaluation in explicit generalized binomial graphs. In: *Proceedings of NIPS*.

Choudhury S, Kapoor A, Ranade G and Dey D (2017b) Adaptive information gathering via imitation learning. In: *Proceedings of RSS*.

Choudhury S, Kapoor A, Ranade G and Dey D (2017c) Learning to gather information via imitation. In: *Proceedings of ICRA*.

Cohen WW and Carvalho VR (2005) Stacked sequential learning. In: *Proceedings of the International Joint Conference on Artificial Intelligence*.

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

Daumé H, Langford J and Marcu D (2009) Search-based structured prediction. *Machine Learning* 75(3): 297–325.

Dellin CM and Srinivasa SS (2016) A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In: *Proceedings of ICAPS*.

Dellin CM, Strabala K, Haynes GC, Stager D and Srinivasa SS (2016) Guided manipulation planning at the DARPA Robotics Challenge Trials. In: *Experimental Robotics* (*Springer Tracts in Advanced Robotics*, vol. 109). Wien: Springer.

Diankov R and Kuffner J (2007) Randomized statistical path planning. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1–6.

Dolgov D, Thrun S, Montemerlo M and Diebel J (2008) Practical search techniques in path planning for autonomous driving. *Proceedings of AAAI*.

Duan Y, Chen X, Houthooft R, Schulman J and Abbeel P (2016) Benchmarking deep reinforcement learning for continuous control. In: *Proceedings of International Conference on Machine Learning*, pp. 1329–1338.

Dubins LE (1957) On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics* 79: 497–516

Dugar V, Choudhury S and Scherer S (2017a) A κite in the wind: Smooth trajectory optimization in a moving reference frame. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 109–116.

Dugar V, Choudhury S and Scherer S (2017b) Smooth trajectory optimization in wind: First results on a full-scale helicopter. In: *AHS International 73rd Annual Forum*, Forth Worth, TX, vol. 1.

Finn C, Levine S and Abbeel P (2016) Guided cost learning: Deep inverse optimal control via policy optimization. In: *International Conference on Machine Learning*, pp. 49–58.

Garrett CR, Kaelbling LP and Lozano-Pérez T (2016) Learning to rank for synthesizing planning heuristics. *Proceedings of the 25th International Joint Conference on Artificial Intelligence*. AAAI Press, pp. 3089–3095.

Golovin D and Krause A (2011) Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research* 42: 427–486.

Golovin D, Krause A and Ray D (2010) Near-optimal Bayesian active learning with noisy observations. In: *Proceedings of NIPS*.

Goodfellow I, Bengio Y and Courville A (2016) *Deep Learning*. Cambridge, MA: MIT Press.

Gupta A, Nagarajan V and Ravi R (2010) Approximation algorithms for optimal decision trees and adaptive TSP problems. In: *International Colloquium on Automata, Languages, and Programming*.

Gupta S, Davidson J, Levine S, Sukthankar R and Malik J (2017) Cognitive mapping and planning for visual navigation. In: *Proceedings of CVPR*.

Hansen N (2016) The CMA evolution strategy: A tutorial. *Preprint arXiv:1604.00772*.

Hausknecht M and Stone P (2015) Deep recurrent Q-learning for partially observable MDPs. In: *2015 AAAI Fall Symposium Series*.

Heng L, Gotovos A, Krause A and Pollefeys M (2015) Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. In: *Proceedings of ICRA*.

Hoffmann J and Nebel B (2001) The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14: 253–302.

Hollinger GA, Englot B, Hover FS, Mitra U and Sukhatme GS (2013) Active planning for underwater inspection and the benefit of adaptivity. *The International Journal of Robotics Research* 32: 3–18.

Hollinger GA, Mitra U and Sukhatme GS (2017) Active classification: Theory and application to underwater inspection. In: *Robotics Research*. New York: Springer, pp. 95–110.

Hollinger GA and Sukhatme GS (2013) Sampling-based motion planning for robotic information gathering. In: *Proceedings of RSS*.

Hsu D, Latcombe JC and Sorkin S (1999) Placing a robot manipulator amid obstacles for optimized execution. In: *Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning, 1999 (ISATP'99)*. IEEE, pp. 280–285.

Ichter B, Harrison J and Pavone M (2017) Learning sampling distributions for robot motion planning. *Preprint arXiv:1709.05448*.

Isler S, Sabzevari R, Delmerico J and Scaramuzza D (2016) An information gain formulation for active volumetric 3D reconstruction. In: *Proceedings of ICRA*.

Iyer RK and Bilmes JA (2013) Submodular optimization with submodular cover and submodular knapsack constraints. In: *Proceedings of NIPS*.

Javdani S, Chen Y, Karbasi A, Krause A, Bagnell JA and Srinivasa S (2014) Near optimal Bayesian active learning for decision making. In: *Proceedings of AISTATS*.

Javdani S, Klingensmith M, Bagnell JA, Pollard N and Srinivasa S (2013) Efficient touch based localization through submodularity. In: *Proceedings of ICRA*.

Javdani S, Srinivasa SS and Bagnell JA (2015) Shared autonomy via hindsight optimization. In: *Proceedings of RSS*.

Jiménez S, De La Rosa T, Fernández S, Fernández F and Borrajo D (2012) A review of machine learning for automated planning. *The Knowledge Engineering Review* 27: 433–467.

Kaelbling LP, Littman ML and Cassandra AR (1998) Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101: 99–134.

Kahn G, Zhang T, Levine S and Abbeel P (2017) Plato: Policy learning using adaptive trajectory optimization. In: *Proceedings of ICRA*.

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

Karkus P, Hsu D and Lee WS (2017) Qmdp-net: Deep learning for planning under partial observability. *Preprint arXiv:1703.06692*.

Kearns MJ, Mansour Y and Ng AY (2000) Approximate planning in large POMDPs via reusable trajectories. In: *Advances in Neural Information Processing Systems*, pp. 1001–1007.

Kelly A and Nagy B (2003) Reactive nonholonomic trajectory generation via parametric optimal control. *The International Journal of Robotics Research* 22(7–8): 583–601.

Kober J, Bagnell JA and Peters J (2013) Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32(11): 1238–1274.

Koval M, Pollard N and Srinivasa S (2014) Pre- and post-contact policy decomposition for planar contact manipulation under uncertainty. In: *Proceedings of RSS*.

Krause A and Golovin D (2012) Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*. Cambridge: Cambridge University Press, pp. 71–104.

Krause A and Guestrin C (2007) Near-optimal observation selection using submodular functions. In: *Proceedings of AAAI*.

Krause A, Leskovec J, Guestrin C, VanBriesen J and Faloutsos C (2008) Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management* 134: 516–526.

Kuffner JJ and LaValle SM (2000) RRT-Connect: An efficient approach to single-query path planning. In: *Proceedings of ICRA*.

Kurniawati H, Hsu D and Lee WS (2008) Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In: *Proceedings of RSS*.

Langford J and Beygelzimer A (2005) Sensitive error correcting output codes. In: *COLT 2005: Learning Theory* (*Lecture Notes in Computer Science*, vol. 3559). New York: Springer, pp. 158–172.

Laumond JP, Sekhavat S and Lamiraux F (1998) Guidelines in nonholonomic motion planning for mobile robots. In: *Robot Motion Planning and Control*. New York: Springer, pp. 1–53.

LaValle SM (2006) *Planning Algorithms*. Cambridge: Cambridge University Press.

LaValle SM and Kuffner JJ (2001) Randomized kinodynamic planning. *The International Journal of Robotics Research* 20: 378–400.

Levine S and Koltun V (2013) Guided policy search. In: *Proceedings of ICML*.

Li H, Liao X and Carin L (2009) Multi-task reinforcement learning in partially observable stochastic environments. *Journal of Machine Learning Research* 10: 1131–1186.

Li J, Monroe W, Ritter A, Galley M, Gao J and Jurafsky D (2016) Deep reinforcement learning for dialogue generation. *Preprint arXiv:1606.01541*.

Liaw A and Wiener M (2002) Classification and regression by randomforest. *R News* 2(3): 18–22.

Likhachev M and Ferguson D (2009) Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research* 28: 933–945.

Lim ZW, Hsu D and Lee WS (2015) Adaptive stochastic optimization: From sets to paths. In: *Proceedings of NIPS*.

Lim ZW, Hsu D and Lee WS (2016) Adaptive informative path planning in metric spaces. *The International Journal of Robotics Research* 35: 585–598.

Littman ML, Cassandra AR and Kaelbling LP (1995) Learning policies for partially observable environments: Scaling up. In: *Proceedings of ICML*.

Littman ML and Sutton RS (2002) Predictive representations of state. In: *Advances in Neural Information Processing Systems*, pp. 1555–1561.

Liu M, Liao X and Carin L (2013) Online expectation maximization for reinforcement learning in POMDPs. In: *Proceedings of IJCAI*.

Madani O, Hanks S and Condon A (2003) On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence* 147(1–2): 5–34.

McAllester DA and Singh S (1999) Approximate planning for factored POMDPs using belief state simplification. In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. San Francisco, CA: Morgan Kaufmann, pp. 409–416.

Mnih V, Kavukcuoglu K, Silver D, et al. (2015) Human-level control through deep reinforcement learning. *Nature* 518: 529–533.

Narayanan V, Aine S and Likhachev M (2015) Improved multi-heuristic A* for searching with uncalibrated heuristics. In: *Eighth Annual Symposium on Combinatorial Search*.

Nelson E and Michael N (2015) Information-theoretic occupancy grid compression for high-speed information-based exploration. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4976–4982.

Ng AY and Jordan M (2000) Pegasus: A policy search method for large MDPs and POMDPs. In: *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. San Francisco, CA: Morgan Kaufmann, pp. 406–415.

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

Papadimitriou CH and Tsitsiklis JN (1987) The complexity of Markov decision processes. *Mathematics of Operations Research* 12(3): 441–450.

Pearl J (1984) *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Boston, MA: Addison-Wesley.

Peters J and Schaal S (2006) Policy gradient methods for robotics. In: *Proceedings of IROS*.

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

Phillips M, Narayanan V, Aine S and Likhachev M (2015) Efficient search with an ensemble of heuristics. In: *Proceedings of IJCAI*.

Pivtoraiko M, Knepper RA and Kelly A (2009) Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics* 26(3): 308–333.

Pohl I (1970) First results on the effect of error in heuristic search. *Machine Intelligence* 5: 219–236.

Ranzato M, Chopra S, Auli M and Zaremba W (2015) Sequence level training with recurrent neural networks. *Preprint arXiv:1511.06732*.

Ratliff ND, Silver D and Bagnell JA (2009) Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots* 27(1): 25–53.

Ross S and Bagnell JA (2014) Reinforcement and imitation learning via interactive no-regret learning. *Prepring arXiv:1406.5979*.

Ross S, Gordon GJ and Bagnell D (2011) A reduction of imitation learning and structured prediction to no-regret online learning. In: *AISTATS*, vol. 1, p. 6.

Ross S, Pineau J, Paquet S and Chaib-Draa B (2008) Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research* 32: 663–704.

Ruml W and Do MB (2007) Best-first utility-guided search. In: *Proceedings of IJCAI*.

Schaul T, Quan J, Antonoglou I and Silver D (2015) Prioritized experience replay. *CoRR* abs/1511.05952.

Silver D, Huang A, Maddison CJ, et al. (2016) Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587): 484–489.

Silver D and Veness J (2010) Monte-Carlo planning in large POMDPs. In: *Proceedings of NIPS*.

Singh A, Krause A, Guestrin C, Kaiser W and Batalin M (2007) Efficient planning of informative paths for multiple robots. In: *Proceedings of IJCAI*.

Singh A, Krause A and Kaiser WJ (2009) Nonmyopic adaptive informative path planning for multiple robots. In: *Proceedings of IJCAI*.

Smith T and Simmons R (2012) Point-based POMDP algorithms: Improved analysis and implementation. *Preprint arXiv:1207.1412*.

Somani A, Ye N, Hsu D and Lee WS (2013) DESPOT: Online POMDP planning with regularization. In: *Proceedings of NIPS*.

Spaan MT and Vlassis N (2005) Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research* 24: 195–220.

Sturm J, Engelhard N, Endres F, Burgard W and Cremers D (2012) A benchmark for the evaluation of RGB-D SLAM systems. In: *Proceedings of IROS*.

Sun W, Venkatraman A, Gordon GJ, Boots B and Bagnell JA (2017) Deeply aggrevated: Differentiable imitation learning for sequential prediction. In: *Proceedings of ICML*.

Sutton RS and Barto AG (1998) *Reinforcement learning: An introduction*, vol. 1. Cambridge, MA: MIT Press.

Tamar A, Thomas G, Zhang T, Levine S and Abbeel P (2016) Learning from the hindsight plan—episodic MPC improvement. *Preprint arXiv:1609.09001*.

Thayer JT, Dionne AJ and Ruml W (2011) Learning inadmissible heuristics during search. In: *Proceedings of ICAPS*.

Theano Development Team (2016) Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688.

Thrun S, Burgard W and Fox D (2005) *Probabilistic robotics*. Cambridge, MA: MIT Press.

Tielman T and Hinton G (2012) Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. In: *COURSERA: Neural Networks for Machine Learning*.

Virseda J, Borrajo D and Alcázar V (2013) Learning heuristic functions for cost-based planning. In *Preprints of the ICAPS'13 PAL Workshop on Planning and Learning*, 2013.

van Hasselt H, Guez A and Silver D (2015) Deep reinforcement learning with double Q-learning. *CoRR* abs/1509.06461.

Venkatraman A, Boots B, Hebert M and Bagnell JA (2014) Data as demonstrator with applications to system identification. In: *ALR Workshop, NIPS*.

Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M and Freitas N (2016) Dueling network architectures for deep reinforcement learning. In: *Proceedings of ICML*.

Watkins CJ and Dayan P (1992) Q-learning. *Machine Learning* 8(3–4): 279–292.

Wilt CM and Ruml W (2015) Building a heuristic for greedy search. In: *Proceedings of the Eighth Annual Symposium on Combinatorial Search*.

Xu Y, Fern A and Yoon S (2009) Learning linear ranking functions for beam search with application to planning. *Journal of Machine Learning Research* 10: 1571–1610.

Xu Y, Fern A and Yoon SW (2007) Discriminative learning of beam-search heuristics for planning. In: *Proceedings of IJCAI*.

Xu Y, Fern A and Yoon SW (2010) Iterative learning of weighted rule sets for greedy search. In: *Proceedings of ICAPS*.

Yoon SW, Fern A and Givan R (2006) Learning heuristic functions from relaxed plans. In: *Proceedings of ICAPS*.

Yu J, Schwager M and Rus D (2014) Correlated orienteering problem and its application to informative path planning for persistent monitoring tasks. In: *Proceedings of IROS*.

Zhang H and Vorobeychik Y (2016) Submodular optimization with routing constraints. In: *AAAI'16 Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pp. 819–825

Zhang T, Kahn G, Levine S and Abbeel P (2016) Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search. In: *Proceedings of ICRA*.

Ziebart BD, Maas AL, Bagnell JA and Dey AK (2008) Maximum entropy inverse reinforcement learning. In: *Proceedings of AAAI*, Chicago, IL, vol. 8, pp. 1433–1438.

Appendix A Proof of Lemma 1

Lemma. *The **offline** imitation of **clairvoyant** oracle (11) is equivalent to sampling **online** a world from the posterior distribution and executing a **hallucinating** oracle as shown in*

$$\hat{\pi} = \arg\max_{\pi\in\Pi} \mathbb{E}_{\substack{t\sim U(1:T),\\ \psi_t\sim P(\psi|\pi,t)}} \left[\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \pi(\psi_t))\right]$$

Proof. We define two loss functions on the policy. Let $\mathcal{L}_1(\pi)$ be the loss function corresponding to clairvoyant oracle, i.e.

$$\mathcal{L}_1(\pi) = \mathbb{E}_{\substack{t\sim U(1:T),\\ s_t,\psi_t\sim P(s,\psi|\pi,t)}} \left[Q^{\pi_{\mathrm{OR}}}_{T-t+1}(s_t, \pi(\psi_t))\right] \tag{25}$$

Let $\mathcal{L}_2(\pi)$ be the loss function corresponding to the hallucinating oracle, i.e.

$$\mathcal{L}_2(\pi) = \mathbb{E}_{\substack{t\sim U(1:T),\\ \psi_t\sim P(\psi|\pi,t)}} \left[\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \pi(\psi_t))\right] \tag{26}$$

Substituting (12) in (26) we have

$$\begin{aligned}
&\mathbb{E}_{\substack{t\sim U(1:T),\\ \psi_t\sim P(\psi|\pi,t)}} \left[\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \pi(\psi_t))\right] \\
&= \mathbb{E}_{\substack{t\sim U(1:T),\\ \psi_t\sim P(\psi|\pi,t)}} \left[\mathbb{E}_{s_t\sim P(s_t|\psi_t)} \left[Q^{\pi_{\mathrm{OR}}}_{T-t+1}(s_t, \pi(\psi_t))\right]\right] \\
&= \mathbb{E}_{t\sim U(1:T)} \left[\sum_{\psi_t, s_t} P(\psi_t|\pi,t)\, P(s_t|\psi_t)\, Q^{\pi_{\mathrm{OR}}}_{T-t+1}(s_t, \pi(\psi_t))\right] \\
&= \mathbb{E}_{t\sim U(1:T)} \left[\sum_{\psi_t, s_t} P(s_t, \psi_t|\pi,t)\, Q^{\pi_{\mathrm{OR}}}_{T-t+1}(s_t, \pi(\psi_t))\right] \\
&= \mathbb{E}_{\substack{t\sim U(1:T),\\ s_t,\psi_t\sim P(s,\psi|\pi,t)}} \left[Q^{\pi_{\mathrm{OR}}}_{T-t+1}(s_t, \pi(\psi_t))\right]
\end{aligned}$$

Hence, $\mathcal{L}_1(\pi) = \mathcal{L}_2(\pi)$. ∎

Appendix B Proof of Theorem 1

We begin with a statement of the *performance difference lemma* that is useful to bound the change in total reward-to-go. This general result bounds the difference in performance of any two policies.

Lemma 2. *Let π and π' be any two policies and denote $\tilde{V}'_t$ and $\tilde{Q}'_t$ be the t-step value function and action value function of policy π' respectively, then*

$$\begin{aligned}
&J(\pi) - J(\pi') \\
&= \sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\pi,t)} \left[\tilde{Q}^{\pi'}_{T-t+1}(\psi_t, \pi(\psi_t)) - \tilde{V}^{\pi'}_{T-t+1}(\psi_t)\right]
\end{aligned}$$

Proof. Let π_t be the policy that executes π in first t timesteps and then switches to π' from $t+1$ to T. We then have $J(\pi) = J(\pi_T)$ and $J(\pi') = J(\pi_0)$. Thus,

$$\begin{aligned}
&J(\pi) - J(\pi') \\
&= \sum_{t=1}^{T} [J(\pi_t) - J(\pi_{t-1})] \\
&= \sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\pi,t)} \left[\tilde{Q}^{\pi'}_{T-t+1}(\psi_t, \pi(\psi_t)) - \tilde{V}^{\pi'}_{T-t+1}(\psi_t)\right]
\end{aligned}$$

∎

We now state the theorem and the proof.

Theorem. FORWARDTRAINING *has the following guarantee*

$$J(\hat{\pi}) \geq J(\tilde{\pi}_{\mathrm{OR}}) - 2T\sqrt{\mathcal{A}\,\varepsilon_{\mathrm{class}}} + T\varepsilon_{\mathrm{or}}$$

where $\varepsilon_{\mathrm{class}}$ is the regression error of the learner, $\varepsilon_{\mathrm{or}}$ is the local oracle suboptimality.

Proof. In FORWARDTRAINING, the distribution of history $P(\psi|\hat{\pi}, t)$ is generated by the learner directly. Let the cost-sensitive classification error $\varepsilon_{\mathrm{cs}}$ be the expected difference in action value selected by the policy and the best action, $\varepsilon_{\mathrm{cs}} =$

$$\mathbb{E}_{\substack{t\sim U(1:T),\\ \psi_t\sim P(\psi|\hat{\pi},t)}} \left[\max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a) - \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \hat{\pi}(\psi_t))\right]$$

We also define the local oracle suboptimality $\varepsilon_{\mathrm{or}}$ being the minimum gap between oracle value and the best action value averaged over all timesteps, i.e.

$$\varepsilon_{\mathrm{or}} = \mathbb{E}_{t\sim U(1:T)} \left[\min_{\psi_t} \left(\max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a) - \tilde{V}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t)\right)\right]$$

This can be non-zero when the oracle is sub-optimal with respect to itself at any timestep. This is true in this setting as there is no guarantee that the hallucinating oracle will pick a locally optimal actions with respect to its own value function. This is true even if the clairvoyant oracle was locally optimal as in the case of search-based planning.

Applying Lemma 2 with $\pi = \hat{\pi}$, $\pi' = \tilde{\pi}_{\mathrm{OR}}$, we have

$$\begin{aligned}
&J(\hat{\pi}) - J(\tilde{\pi}_{\mathrm{OR}}) \\
&= \sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\hat{\pi},t)} \left[\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \hat{\pi}(\psi_t)) - \tilde{V}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t)\right] \\
&= \sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\hat{\pi},t)} [\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \hat{\pi}(\psi_t)) \\
&- \max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a)] \\
&+ \sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\hat{\pi},t)} \left[\max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a) - \tilde{V}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t)\right] \\
&\geq -T\varepsilon_{\mathrm{cs}} + \\
&\sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\hat{\pi},t)} \left[\max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a) - \tilde{V}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t)\right] \\
&\geq -T\varepsilon_{\mathrm{cs}} + T\varepsilon_{\mathrm{or}}
\end{aligned}$$

Hence, we have the performance bound

$$J\left(\hat{\pi}\right) \geq J\left(\tilde{\pi}_{\mathrm{OR}}\right) - T\varepsilon_{\mathrm{cs}} + T\varepsilon_{\mathrm{or}}$$

Interestingly, note that if $\varepsilon_{\mathrm{or}} \geq \varepsilon_{\mathrm{cs}}$, we would be guaranteed to perform *better* than the hallucinating oracle.

As we reduce cost-sensitive classification to regression by uniformly sampling actions, we can express $\varepsilon_{\mathrm{cs}}$ in terms of the regression error $\varepsilon_{\mathrm{class}}$ using the reduction bound from Langford and Beygelzimer (2005):

$$\varepsilon_{\mathrm{cs}} \leq 2\sqrt{|\mathcal{A}|\ \varepsilon_{\mathrm{class}}}$$

Hence, we have the performance bound

$$J\left(\hat{\pi}\right) \geq J\left(\tilde{\pi}_{\mathrm{OR}}\right) - 2T\sqrt{|\mathcal{A}|\ \varepsilon_{\mathrm{class}}} + T\varepsilon_{\mathrm{or}}$$

■

Appendix C Proof of Theorem 2

We follow the analysis of Ross and Bagnell (2014) with two main differences:

1. Ross and Bagnell (2014) examined an MDP on states, we translate that to an MDP on history;
2. Ross and Bagnell (2014) considered one-step cost minimization, we consider one-step reward maximization.

We borrow a couple important lemmas from Ross and Bagnell (2014).

Lemma 3. *Let P and Q be any two distributions over x, let $f(x)$ be a bounded function with range r. We then have*

$$\left|\mathbb{E}_{x\sim P}\left[f(x)\right] - \mathbb{E}_{x\sim Q}\left[f(x)\right]\right| \leq \frac{r}{2}\left|\left|P - Q\right|\right|_1$$

Lemma 4. *Let $P(\psi|\pi_{\mathrm{mix},i})$ be the distribution of history encountered by the mixture policy over all timesteps and $P(\psi|\hat{\pi}_i)$ be the distribution encountered by the learner. We have*

$$\left|\left|P(\psi|\pi_{\mathrm{mix},i}) - P(\psi|\hat{\pi}_i)\right|\right|_1 \leq 2\min(1, T\beta_i)$$

We now state the theorem we wish to prove.

Theorem. *There are N iterations of* AGGREVATE *collecting m regression examples per iteration, which guarantees that with probability at least $1-\delta$*

$$\begin{aligned} J\left(\hat{\pi}\right) \geq & J\left(\tilde{\pi}_{\mathrm{OR}}\right) \\ & - 2T\sqrt{|\mathcal{A}|\left(\varepsilon_{\mathrm{class}} + \varepsilon_{\mathrm{reg}} + \mathcal{O}\left(\sqrt{\log 1/\delta / Nm}\right)\right)} \\ & - \mathcal{O}\left(\frac{R\ T\log T}{N}\right) + T\varepsilon_{\mathrm{or}} \end{aligned}$$

where $\varepsilon_{\mathrm{class}}$ is the empirical regression regret of the best regressor in the regression class on the aggregated dataset, $\varepsilon_{\mathrm{reg}}$ is the empirical online learning average regret on the sequence of training examples, R is the range of oracle action value, and $\varepsilon_{\mathrm{or}}$ is the local oracle suboptimality.

Proof. We first define the local oracle suboptimality $\varepsilon_{\mathrm{or}}$ as in Appendix 8.4

$$\varepsilon_{\mathrm{or}} = \mathbb{E}_{t\sim U(1:T)}\left[\min_{\psi_t}\left(\max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a) - \tilde{V}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t)\right)\right]$$

We define the average cost-sensitive classification error $\varepsilon_{\mathrm{cs}}$

$$\varepsilon_{\mathrm{cs}} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\substack{t\sim U(1:T),\\ \psi_t\sim P(\psi|\pi_{\mathrm{mix},i},t)}}\left[\max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a) - \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \hat{\pi}(\psi_t))\right]$$

Applying the performance difference lemma in Lemma 2 with $\pi = \hat{\pi}_i$, $\pi' = \tilde{\pi}_{\mathrm{OR}}$, we have

$$\begin{aligned} & J\left(\hat{\pi}_i\right) - J\left(\tilde{\pi}_{\mathrm{OR}}\right) \\ & = \sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\hat{\pi}_i,t)}\left[\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \hat{\pi}_i(\psi_t)) - \tilde{V}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t)\right] \\ & = \sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\hat{\pi}_i,t)}[\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \hat{\pi}_i(\psi_t)) \\ & - \max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a)] \\ & + \sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\hat{\pi}_i,t)}\left[\max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a) - \tilde{V}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t)\right] \\ & \geq \sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\hat{\pi}_i,t)}[\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \hat{\pi}_i(\psi_t)) \\ & - \max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a)] \\ & + T\varepsilon_{\mathrm{or}} \end{aligned}$$

We define the range R of the maximum difference between the best and worst action value function of the oracle:

$$R = \max_{t,\psi_t}\left|\max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a) - \min_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a)\right|$$

We can then apply Lemmas 3 and 4 with $P(\psi|\hat{\pi}_i, t)$ and $P(\psi|\pi_{\mathrm{mix},i}, t)$ to obtain

$$\begin{aligned} & J\left(\hat{\pi}_i\right) - J\left(\tilde{\pi}_{\mathrm{OR}}\right) \\ & \geq \sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\pi_{\mathrm{mix},i},t)}[\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \hat{\pi}_i(\psi_t)) \\ & - \max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a)] \\ & - \frac{R}{2}\sum_{t=1}^{T}\left|\left|P(\psi|\pi_{\mathrm{mix},i}, t) - P(\psi|\hat{\pi}_i, t)\right|\right|_1 + T\varepsilon_{\mathrm{or}} \\ & \geq \sum_{t=1}^{T} \mathbb{E}_{\psi_t\sim P(\psi|\pi_{\mathrm{mix},i},t)}[\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \hat{\pi}_i(\psi_t)) \\ & - \max_{a\in\mathcal{A}} \tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, a)] \\ & - R\ T\ \min(1, T\beta_i) + T\varepsilon_{\mathrm{or}} \end{aligned}$$

If we now wish to bound the performance of the average learner over N iterations

$$\begin{aligned} & J\left(\hat{\pi}_{\text{avg}}\right) - J\left(\tilde{\pi}_{\text{OR}}\right) \\ &= \frac{1}{N}\sum_{i=1}^{N}\left|J\left(\hat{\pi}_i\right) - J\left(\tilde{\pi}_{\text{OR}}\right)\right| \\ &\geq -T\varepsilon_{\text{cs}} - \frac{R\,T}{N}\sum_{i=1}^{N}\min(1, T\beta_i) + T\varepsilon_{\text{or}} \\ &\geq -T\varepsilon_{\text{cs}} - \frac{R\,T}{N}\frac{\log(T)+2}{\alpha} + T\varepsilon_{\text{or}} \end{aligned}$$

where the last inequality follows from Ross and Bagnell (2014) after setting $\beta_i = (1-\alpha)^{i-1}$.

To bound ε_{cs}, we need to define two terms: $\varepsilon_{\text{class}}$, the empirical regression regret of the best regressor in the regression class on the aggregated dataset; and ε_{reg}, the empirical online learning average regret on the sequence of training examples. We then use the following result from Ross and Bagnell (2014):

$$\varepsilon_{\text{cs}} \leq 2\sqrt{|\mathcal{A}|\left(\varepsilon_{\text{class}} + \varepsilon_{\text{reg}} + \mathcal{O}\left(\sqrt{\log 1/\delta / Nm}\right)\right)}$$

with probability $1-\delta$.

Also note that the performance of the best policy in the sequence $\hat{\pi}$ is better than the average learner, i.e. $J\left(\hat{\pi}\right) \geq J\left(\hat{\pi}_{\text{avg}}\right)$.

This results in the following bound for AGGREVATE with probability $1-\delta$

$$\begin{aligned} J\left(\hat{\pi}\right) \geq & J\left(\tilde{\pi}_{\text{OR}}\right) \\ & - 2T\sqrt{|\mathcal{A}|\left(\varepsilon_{\text{class}} + \varepsilon_{\text{reg}} + \mathcal{O}\left(\sqrt{\log 1/\delta / Nm}\right)\right)} \\ & - \mathcal{O}\left(\frac{R\,T\log T}{N}\right) + T\varepsilon_{\text{or}} \end{aligned}$$

■

Appendix D Proof of Theorem 3

Theorem. *There are N iterations of* AGGREVATE *with clairvoyant one-step-reward collecting m regression examples per iteration, which guarantees that with probability at least $1-\delta$*

$$\begin{aligned} J\left(\hat{\pi}\right) \geq & \left(1 - \frac{1}{e}\right) J\left(\pi^*\right) \\ & - 2T\sqrt{|\mathcal{A}|\left(\varepsilon_{\text{class}} + \varepsilon_{\text{reg}} + \mathcal{O}\left(\sqrt{\log 1/\delta / Nm}\right)\right)} \\ & - \mathcal{O}\left(\frac{R\,T\log T}{N}\right) \end{aligned}$$

where $\varepsilon_{\text{class}}$ is the empirical regression regret of the best regressor in the regression class on the aggregated dataset, ε_{reg} is the empirical online learning average regret on the sequence of training examples, and R is the maximum range of one-step-reward.

Proof. We use an important result from Golovin and Krause (2011) about the near-optimality properties of greedy maximization of an adaptive monotone and adaptive submodular set function. We define a greedy policy.

The greedy algorithm selects a node to visit that has the highest expected marginal gain under the conditional distribution of world maps given the history. If the history of vertices visited and measurements received are where $\psi = \{v_i\}_{i=1}^{t}, \{y_i\}_{i=1}^{t}$, the greedy algorithm $\pi_{\text{GR}}(\psi_t)$ selects node to visit v_{t+1} with the highest expected marginal gain

$$v_{t+1} = \arg\max_{v\in\mathcal{V}} \mathbb{E}_{\phi\sim P(\phi|\psi_t)}\left[\Delta_{\mathcal{F}}\left(v|\{v_i\}_{i=1}^{t}, \phi\right)\right] \tag{27}$$

Golovin and Krause (2011) show that the greedy algorithm π_{GR} has the following guarantee.

Lemma 5. *If $\mathcal{F}$ is adaptive monotone and adaptive submodular with respect to $P(\phi)$ and π_{GR} is a greedy policy, then for all policies π^* we have*

$$\mathbb{E}_{\phi\sim P(\phi)}\left[\mathcal{F}\left(\pi_{\text{GR}}, \phi\right)\right] \geq \left(1 - \frac{1}{e}\right)\mathbb{E}_{\phi\sim P(\phi)}\left[\mathcal{F}\left(\pi^*, \phi\right)\right]$$

We note that for the clairvoyant one-step-reward oracle is defined such that

$$\begin{aligned} \tilde{\pi}_{\text{OR}}(\psi_t) &= \arg\max_{a\in\mathcal{A}} \tilde{Q}_{T-t+1}^{\tilde{\pi}_{\text{OR}}}(\psi_t, a) \\ &= \arg\max_{v_{t+1}\in\mathcal{V}} \mathbb{E}_{\phi\sim P(\phi|\psi)}\left[\Delta_{\mathcal{F}}\left(v_{t+1}|\{v_i\}_{i=1}^{t}, \phi\right)\right] \end{aligned} \tag{28}$$

where the second inequality uses Definition 3 along with Definition 2. Hence, $\tilde{\pi}_{\text{OR}} = \pi_{\text{GR}}$. In addition, the local suboptimality is $\varepsilon_{\text{or}} = 0$ since the oracle selects actions that maximize one-step-reward. Finally, the range R is that of the one step reward.

Hence, applying these terms along with Lemma 5 in Theorem 2, we have

$$\begin{aligned} J\left(\hat{\pi}\right) \geq & \left(1 - \frac{1}{e}\right) J\left(\pi^*\right) \\ & - 2T\sqrt{|\mathcal{A}|\left(\varepsilon_{\text{class}} + \varepsilon_{\text{reg}} + \mathcal{O}\left(\sqrt{\log 1/\delta / Nm}\right)\right)} \\ & - \mathcal{O}\left(\frac{R\,T\log T}{N}\right) \end{aligned}$$

■

Appendix E Machine learning baselines for search-based planning

E.1 Supervised learning (behavior cloning)

The supervised learning algorithm is identical to SAIL with the key difference that roll-outs are made with π_{OR} and not $\pi_{\text{mix},i}$. This is equivalent to setting the mixing parameter $\beta = 1$ across all environments. For completeness, we present the algorithm in Algorithm 6.

We use $m = 600$ for all the environments. The network architecture and hyper-parameters used are the same as SAIL.

Algorithm 6 Supervised Learning ($P(\phi), P(v_s, v_g)$)

Initialize $\mathcal{D} \leftarrow \emptyset$
Collect datapoints as follows:
for $i = 1, \ldots, m$ **do**
  Initialize sub-dataset $\mathcal{D}_i \leftarrow \emptyset$
  Sample $\phi \sim P(\phi)$
  Sample $\left(v_s, v_g\right) \sim P(v_s)$
  Invoke clairvoyant oracle planner to compute $Q^{\mathrm{OR}}(v, \phi)\ \forall v \in \mathcal{V}$
  Roll-out a new search with π_{OR}
  At each timestep t pick a random action a_t to get corresponding (v, s_t)
  Query oracle for $Q^{\mathrm{OR}}(v, \phi)$
  $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \left\langle v, s_t, Q^{\mathrm{OR}}(v, \phi) \right\rangle$
  Continue roll-out with π_{OR} till end of episode.
Append to c.s. classification data $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
Train on $\mathcal{D}$ to get $\hat{\pi}$
Return $\hat{\pi}$

Algorithm 7 Q-Learning ($P(\phi), P(v_s, v_g), k$)

Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π
for $i = 1, \ldots, N$ **do**
  Initialize sub-dataset $\mathcal{D}_i \leftarrow \emptyset$
  Let mixture policy be $\pi_{\mathrm{mix},i} = \epsilon$-greedy on $\hat{\pi}_i$ with ϵ_i
  Collect mk datapoints as follows:
  for $j = 1, \ldots, m$ **do**
    Sample $\phi \sim P(\phi)$
    Sample $\left(v_s, v_g\right) \sim P(v_s)$
    Sample uniformly k timesteps $\{t_1, t_2, \ldots, t_k\}$ where each $t_i \in \{1, \ldots, T\}$
    Roll-out a new search with $\pi_{\mathrm{mix},i}$
    At each $t \in \{t_1, t_2, \ldots, t_k\}$, $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \langle v, s_t, C, v_{t+1} \rangle$
    Continue roll-out with $\pi_{\mathrm{mix},i}$ until the end of the episode.
  Append to dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
  Train learner by minimizing TD error on $\mathcal{D}$ to get $\hat{\pi}_{i+1}$
Return Best $\hat{\pi}$ on validation

E.2 Q-learning with function approximation

We use an episodic implementation of the Q-learning algorithm which collects data in an iteration-wise manner similar to SAIL. The learner is trained on the aggregated dataset across all iterations by regressing to the TD error. The aggregated dataset $\mathcal{D}$ effectively acts as an experience replay buffer to which helps in stabilizing learning when using neural network function approximation as has been suggested in recent work (Mnih et al., 2015). However, we do not use a target network or any other extensions over the original Q-learning algorithm in our baselines (van Hasselt et al., 2015; Schaul et al., 2015). We also use only a single observation to take decisions and not a history length of past h observations for a fair comparison with SAIL that also uses a single observation. Algorithm 7 describes the training procedure for the Q-learning baseline.

Here C is the one-step cost which is 1 for every expansion until goal is added to the open list. We use $k = 100$ and $\epsilon_0 = 0.9$. Epsilon is decayed after every iteration in an exponential manner. Network architecture and parameters are kept the same as SAIL.

E.3 Cross-entropy method

We use a cross-entropy method (CEM) as a derivative-free optimization method for training (Goodfellow et al., 2016). At each iteration of the algorithm we sample $\mathrm{batch}_{\mathrm{size}} = 40$ set of parameters from a Gaussian distribution. Each parameter is used to roll-out a policy on five environments each and the total cost is collected. The total cost (number of expansions) is used as the fitness function and the best performing, $n_{elite} = 20\%$ of the parameters are selected. These elite parameters are then used to create a new Gaussian distribution (using sample mean and standard deviation) for the next iteration. At the end of all iterations, the best-performing policy on a set of held-out states is returned. For this baseline, we use a simpler neural network architecture with one hidden layer of 100 units and ReLu activation.

Appendix F Representation for search-based planning

To overcome the changing sizes of the observation and action spaces in our setting, we use insight from the motion-planning literature and represent an entire search state in terms of closest nodes in $\mathcal{O}$ to a set of pre-defined *attractor states* and *attractor paths*. Attractor states are manually defined states that can be thought of as landmarks trying to pull the search cloud in different directions. Such states can be useful in pulling the search out of local minima such as a bugtrap or they could be strategic orientations of the robot or an object the robot is trying to manipulate that lead to faster solutions (Aine et al., 2016). Attractor paths on the other hand are solutions to a small subset of environments from the training dataset. In many episodic tasks, where the structure of the environment does not change drastically between planning iterations, such *path-reuse* can be very useful in finding solutions faster (Phillips et al., 2012). The planning algorithm is built into the environment, and the agent only receives as an observation the nodes in the open list closest to each attractor paths/states. At each iteration then, the action that the agent performs is to select a node from the observation to expand.

Although this is a generic framework that can be applied to many different problems, we chose not to use it for this work. The reason for this choice was that in this paper, our aim was to build the foundation for learning graph search heuristics as sequential decision-making problem and clearly demonstrate the efficacy of the IL paradigm

in this domain. We found that using attractor paths/states would distract from the effectiveness of SAIL and also make learning easier for other baseline methods.

In our final experiments, we instead featurize every pair (v, s) using simple information based on the search tree and the environment uncovered up until that point.

Appendix G Analyzing the time complexity of SaIL

The computational bottleneck in SAIL is the `Select` function, which requires estimating the Q-value for every node in the open list $\mathcal{O}$. Contrast this with something like Dijkstra's algorithm which selects a node to expand in very little time, but wastes a lot of computation in excessively expanding nodes and evaluating edges. In order to analyze the usefulness of SAIL in terms of computational gains, we make the following assumptions. First, we assume that the computational cost of calculating the Q-value of a single node (including feature calculation and forward pass through function approximator) is equal to the computational cost of `Expand` function (involves checking all edges coming out of a node for collision and calculating edge costs). This is in reality a very conservative approximation as in many high-dimensional planning problems, collision checking is way more computationally demanding as it requires expensive geometric intersection computations. We also ignore the computational cost of re-ordering the priority-queue whenever a node is popped which means Dijkstra's algorithm can select a node to expand in $O(1)$. Given a graph with cardinality k, we obtain the following time complexities for SAIL and Dijkstra's algorithm.

SAIL *test time complexity*

Assume SAIL did A expansions before it found a solution starting from an empty $\mathcal{O}$. In addition, assume that states are never removed from $\mathcal{O}$. Total `Select` complexity: $O(k + 2k + 3k + \ldots + Ak) = O\left(kA^2\right)$. Total expansion complexity: $O(\sum_{i=1}^{A} k) = O(Ak)$. From this we obtain the total complexity of SAIL to be $O\left(kA^2\right)$.

Dijkstra's test time complexity

Assume Dijkstra's algorithm does B expansions before finding a path. As mentioned earlier, we assume that priority-queue reordering can be achieved in constant time. Total `Select` complexity: $O(1)$. Total expansion complexity: $O(kB)$. From this we obtain total complexity for Dijkstra's algorithm to be $O(kB)$.

From the above analysis, for SAIL to have less overall computational complexity than uninformed search we require the following condition to be satisfied:

$$A^2 < B \tag{29}$$

Thus, SAIL must obtain a squared reduction in total number of expansions for it to be computationally better than uninformed search. We argue that this strengthens the case for using SAIL in higher-dimensional search graphs as in uninformed search expands a very large number of nodes as the total number of graph nodes increases.