Approximately-Optimal Queries for Planning in Reward-Uncertain Markov Decision Processes

Shun Zhang, Edmund Durfee, and **Satinder Singh**
Computer Science and Engineering
University of Michigan
shunzh,durfee,baveja@umich.edu

Abstract

When planning actions to take on behalf of its human operator, a robot might be uncertain about its operator's reward function. We address the problem of how the robot should formulate an (approximately) optimal query to pose to the operator, given how its uncertainty affects which policies it should plan to pursue. We explain how a robot whose queries ask the operator to choose the best from among k choices can, without loss of optimality, restrict consideration to choices only over alternative *policies*. Further, we present a method for constructing an approximately-optimal policy query that enjoys a performance bound, where the method need not enumerate all policies. Finally, because queries posed to the operator of a robotic system are often expressed in terms of preferences over *trajectories* rather than policies, we show how our constructed policy query can be *projected* into the space of trajectory queries. Our empirical results demonstrate that our projection technique can outperform other known techniques for choosing trajectory queries, particularly when the number of trajectories the operator is asked to compare is small.

Introduction

When a computational agent (which we refer to as "the robot") plans actions to take in its environment, it needs knowledge of the goals and preferences of the person who deployed it ("the operator"). Circumstances can arise where the robot can be uncertain about the operator's goals or preferences, such as when the operator omits preferences over unlikely situations that end up arising. Since the operator may be unaware of when the robot faces significant uncertainty, we consider the question of how a decision-theoretic robot can use what it knows, about what it might plan to do and what the operator's preferences for those plans might be, to formulate a query that (approximately) optimally improves its model of the operator's preferences.

Our strategy directs the robot to query so as to resolve uncertainty that most affects its plans, rather than maximally reducing its overall uncertainty. To avoid distractions, in this paper we assume uncertainty does not arise in other aspects of the problem, and specifically that the operator is certain of her true preferences, and certain about how to answer the query to correctly reflect her preferences. To bound the reasoning demands on the operator for answering a query, we assume that the query takes the form of a "multiple choice" question, where the operator is asked to select the most preferred choice from k possible responses. Our approach does not restrict k (other than that it is finite), but our empirical results suggest that our approach is particularly good for smaller values of k, which arguably correspond to "easier" queries to answer.

Note that this form of interaction to elicit operator preferences differs from learning from demonstration and inverse reinforcement learning (Abbeel and Ng 2004), where the operator initiates interaction by demonstrating a preferable behavior for the robot to learn. The operator may not know the robot's planning/learning algorithm, so she may not know when the robot needs teaching signals (Torrey and Taylor 2013). Knox and Stone (2009) consider a setting where the operator gives positive or negative feedback during a robot's demonstration, but again the responsibility is on the operator to decide which behaviors to reinforce.

Our work instead draws inspiration from prior research on a robot querying its operator to elicit preferences, e.g., (Wilson, Fern, and Tadepalli 2012; Zhang and Stone 2015; Akrour, Schoenauer, and Sebag 2012). Like past work, we consider queries comprised of trajectories, where the operator expresses her preference between alternative sequences of actions over states. Past work, however, assumed the robot could ask as many queries as it needed to to sufficiently resolve its uncertainty. In contrast, here we assume that for time and/or attentional reasons the robot is limited to a single query (Viappiani and Boutilier 2010; Cohn, Singh, and Durfee 2014) with a finite number k of responses, and so must choose that query wisely.

Our contributions are twofold. First, drawing on previous results in recommendation settings (Viappiani and Boutilier 2010; Cohn, Singh, and Durfee 2014), we observe that, without loss of optimality, we can restrict the robot to only considering k-response queries that ask the operator to choose between policies. The previous results also specified a greedy construction method for finding an approximately-optimal query whose computational complexity is linear in the number of choices (recommendations) to consider offering. In our setting, however, the number of choices (policies) is intractably large, making it impractical to apply the previous methods directly. Our first major contribution is thus a new method to greedily construct an approximately optimal policy query without enumerating all policies.

Although the optimal k-response policy query is always an optimal k-response query, past work on human-robot interaction has not posed such policy queries directly to operators, perhaps because a policy over the entire state space is too large and complex for the operator to directly reason about. Previous work has instead offered operators choices between partial specifications of policies, such as between alternative fixed-length trajectories from a chosen state. (Essentially: "Starting here, which of these short plans most resembles what you would want to do?") Our second contribution is to provide a process the robot can use to efficiently find a k-response trajectory query based on the approximately-optimal policy query, and to emprically demonstrate that, under certain conditions, it can outperform prior methods for finding trajectory queries (Wilson, Fern, and Tadepalli 2012).

Background

Preference Elicitation in HRI

Preference elicitation in human-robot interaction (HRI) typically involves two agents, a human operator who knows the true reward function r^*, and a robot who takes actions on behalf of the operator. The robot knows a set of possible reward candidates $\mathcal{R}$ that includes r^*, and a probability distribution ψ over the reward candidates regarding which is r^*. The robot has a space of decisions, $D = \{d_1, d_2, \dots, d_m\}$, that it is choosing between; in our setting, each decision is a candidate plan (policy) to pursue. The utility of a decision $d \in D$ under a reward candidate $r \in \mathcal{R}$ is denoted by V_r^d. The robot wants to pick the decision with the highest utility given r^*, but since it only has probabilistic knowledge ψ about which reward candidate is r^*, it maximizes the expected utility given ψ. Communication between the operator and robot has the robot offering a choice over some finite number k decisions, and the operator responding with the decision with the largest utility value under her true reward function. Usually, $|D|$ is too large to offer all possible choices, so the challenge is to select a (often much) smaller subset of k decisions to best elicit the operator's preferences.

Formally, a k-response decision query q is a cardinality k set of decisions, $q = \{d_1, d_2, \dots, d_k\} \in D^k$. Following accepted notation (Viappiani and Boutilier 2010), $q \rightsquigarrow d$, where $d \in q$, denotes the event that the operator selects d from q. $q \rightsquigarrow d \implies V_{r^*}^d \geq V_{r^*}^{d'}, \forall d' \in q$. If multiple decisions attain the optimal V_{r^*} value, the operator returns the lowest indexed (Cohn 2016).

The goal of the robot is to find a q that maximizes its Expected Posterior Utility (EPU), defined as follows.

$$EPU(q, \psi) = \sum_{d \in q} \mathbb{P}[q \rightsquigarrow d; \psi] V^*_{q \rightsquigarrow d; \psi} \qquad (1)$$

where $V^*_{q \rightsquigarrow d;\psi} = \max_{d' \in D} V^{d'}_{q \rightsquigarrow d;\psi}$, and $q \rightsquigarrow d; \psi$ is the posterior belief about the reward function after observing $q \rightsquigarrow d$, with the prior belief ψ. We define how to compute it later. A (k-response) decision query that optimizes the function above is called the *optimal (k-response) decision query*. The optimal decision query maximizes the utility of the optimal decision after knowing the query response in expectation. Or equivalently, this maximizes the Expected Value of Information (EVOI) (Cohn, Singh, and Durfee 2014), where $EVOI(q, \psi) = EPU(q, \psi) - V_\psi^*$ is the utility increase in expectation by asking q. Queries that maximize EPU also maximize $EVOI$, since V_ψ^* is independent of q.

It is difficult to find a q that maximizes EPU directly since there is an extra $\max$ function inside EPU (in V_ψ^*). Viapianni and Boutilier (2010) suggest that we can optimize a different objective function, Expected Utility of Selection (EUS), defined as follows.

$$EUS(q, \psi) = \sum_{d \in q} \mathbb{P}[q \rightsquigarrow d; \psi] V^d_{q \rightsquigarrow d; \psi} \qquad (2)$$

The query q that maximizes the function above is called the *optimal recommendation set*. The difference is that EUS evaluates the response d under that posterior belief, rather than finding the optimal decision under the posterior belief. Theorem 2 in (Viappiani and Boutilier 2010) says if q maximizes $EUS(q, \psi)$, it also maximizes $EPU(q, \psi)$. We maximize EUS in this paper.

Reward-Uncertain MDPs

In our setting, the robot's decisions are over action policies for a reward-uncertain MDP, which is a tuple, $\langle S, A, T, (\mathcal{R}, \psi), s_0 \rangle$, with state space S and action space A. T is the transition function, where $T(s, a, s')$ is the probability of reaching state s' by taking action a in s. $\mathcal{R} = \{r_1, \dots, r_n\}$ is a set of candidate reward functions and ψ is the belief distribution over the candidates. s_0 is the initial state. As in the general preference elicitation setting, r^* is only known to the operator and the robot only knows $\mathcal{R}$ and ψ.

The value of a policy π under reward function r is defined as $V_r^\pi(s) = \sum_{t=0}^T \mathbb{E}[r(s_t, a_t) | s_0 = s, s_t, a_t \sim \pi]$. When the initial state is clear from the context, we abbreviate $V_r^\pi(s)$ as V_r^π. The optimal policy π^* for the reward function r is defined as the policy that obtains the largest value, $\pi^* = \arg\max_\pi V_r^\pi$.

The optimal policy under reward belief ψ is the same as the optimal policy under the reward function $\bar{r}_\psi$, where $\bar{r}_\psi$ is the mean reward under ψ, defined as $\bar{r}_\psi(s, a) = \sum_{r \in \mathcal{R}} \mathbb{P}[r; \psi] r(s, a), \forall s \in S, a \in A$ (Theorem 3 in (Ramachandran and Amir 2007)).

Although there are popular methods like value iteration that finds the optimal policy, we consider a linear programming formulation (de Farias and Van Roy 2003) in this paper. The following optimization problem finds the occupancy measure of the optimal policy, where occupancy measure, $x(s, a)$, is the expected number of times state s will be reached and action a will be taken in state s.

$$\max_x \sum_{s,a} x(s,a)\bar{r}_\psi(s,a) \tag{3}$$
$$\text{s.t.} \sum_{a'} x(s',a') = \sum_{s,a} x(s,a)T(s,a,s') + \delta(s',s_0), \forall s'$$
$$x(s,a) \geq 0, \forall s,a$$

where $\delta(s,s') = 1$ if $s = s'$. Otherwise $\delta(s,s') = 0$.

Querying in Reward-Uncertain MDPs

In the literature, the types of queries asked by robots to their operators have included action queries (Cohn, Durfee, and Singh 2011) and trajectory queries (Wilson, Fern, and Tadepalli 2012). The robot updates its reward belief after receiving the response. It computes the posterior probability of a reward candidate r being the operator's true reward function, $\mathbb{P}[r|q \rightsquigarrow j;\psi]$, based on Bayes' rule:

$$\mathbb{P}[r|q \rightsquigarrow j;\psi] \sim \mathbb{P}[q \rightsquigarrow j|r]\mathbb{P}[r;\psi] \tag{4}$$

Policy Queries

The preceding section provides a straightforward but expensive strategy for finding an optimal query: For every query that could conceivably be asked, compute its EPU (Eq. 1), which for each query involves computing the probability of each response, the posterior beliefs due to the response, and the optimal policy to follow given those beliefs. Then ask the query with the highest EPU. We now consider ways to find (approximately) optimal queries with much less computation.

A fundamental result that we use is that, in a general preference-elicitation setting, a robot can, without loss of optimality, restrict the choices offered in k-response queries to only the *decisions* it can actually act on (Cohn, Singh, and Durfee 2014). Since the robot is deciding among policies, this means that, in reward-uncertain MDPs, a robot can focus only on k-response *policy* queries. Formally, a policy query, $q = \{\pi_1, \ldots, \pi_k\}$, is a set of policies, and the response is such that $q \rightsquigarrow \pi \implies V^{\pi}_{r^*} \geq V^{\pi'}_{r^*}, \forall \pi' \in q$.

We reapply the general result for decision queries in (Cohn, Singh, and Durfee 2014) to assert the following theorem about policy queries.

Theorem 1. *(adapted from (Cohn, Singh, and Durfee 2014)) For a reward-uncertain MDP, the optimal k-response policy query is the optimal k-response query.*

$$\max_{q \in \Pi^k} EPU(q,\psi) = \max_{q \in Q_k} EPU(q,\psi) \tag{5}$$

for any initial reward belief ψ. Q_k is the space of all k-response queries, and Π is the space of all policies.

We do not repeat their proof of Theorem 1 here, but the intuition is as follows. Suppose $q^* = \{c_1, \ldots, c_k\}$ is an optimal query. We can construct a set of k policies, where $\pi_i = \arg\max_\pi V^{\pi}_{q^* \rightsquigarrow c_i;\psi}$. That is, π_i is the optimal policy under the posterior belief of $q^* \rightsquigarrow c_i;\psi$. Let $q = \{\pi_1, \ldots, \pi_k\}$. Then q and q^* have equal EPU.

Although the robot can, without loss of optimality, limit its query search to k-response policy queries, there are still many such queries because the space of k-ary policy query candidates is of size $\binom{|\Pi|}{k}$. Given the exponential number of policies, we need a more efficient way to find an approximately optimal policy query.

Greedy Construction of Policy Queries

Since optimizing EPU directly is generally difficult, we first consider the EUS of a policy query.

$$\begin{aligned}
EUS(q,\psi) &= \sum_{\pi \in q} \mathbb{P}[q \rightsquigarrow \pi;\psi] V^{\pi}_{q \rightsquigarrow \pi;\psi} \\
&= \sum_{\pi \in q} \mathbb{P}[q \rightsquigarrow \pi;\psi] \sum_{r \in \mathcal{R}} \mathbb{P}[r|q \rightsquigarrow \pi;\psi] V^{\pi}_r \\
&\qquad \text{(linearity of value function of a fixed policy)} \\
&= \sum_{r \in \mathcal{R}} \sum_{\pi \in q} \mathbb{P}[r, q \rightsquigarrow \pi;\psi] V^{\pi}_r \\
&= \sum_{r \in \mathcal{R}} \mathbb{P}[r;\psi] \max_{\pi \in q} V^{\pi}_r \\
&\qquad (\mathbb{P}[r, q \rightsquigarrow \pi;\psi] > 0 \iff \pi = \arg\max\nolimits_{\pi \in q} V^{\pi}_r)
\end{aligned}$$

This describes EUS for a set of policies as the expectation of optimal values (achieved by policies in the query) over all possible reward candidates.

In a general preference elicitation setting, Viappiani and Boutilier (2010) proposed a greedy construction method for finding an approximately optimal query. Instead of finding the k elements in the query altogether, the algorithm incrementally adds decisions to the query. Concretely, in the first iteration, $q_1 = \{d^*_\psi\}$ contains the optimal decision under prior belief ψ. In later iteration i, where $i = 2\ldots,k$, the algorithm finds $q_i = q_{i-1} \cup \{d_i\}$ where $d_i = \arg\max_{d \in D} EUS(q_{i-1} \cup \{d\}, \psi)$. This algorithm performs k passes over the decision space, so its computational complexity is $O(|D|k)$. In our setting, this is $O(|\Pi|k)$, which is impractical given the size of Π.

We now define our variation of this process for reward-uncertain MDPs that avoids examining all of Π. Initially, $q_1 = \{\pi^*_\psi\}$, containing the optimal policy under ψ. In iteration i, $q_i = q_{i-1} \cup \{\pi_i\}$ where $\pi_i = \arg\max_\pi EUS(q_{i-1} \cup \{\pi\}, \psi)$. Consider the difference between $EUS(q_{i-1} \cup \{\pi\}, \psi)$ and $EUS(q_{i-1}, \psi)$.

$$\begin{aligned}
&\max_\pi EUS(q_{i-1} \cup \{\pi\}, \psi) - EUS(q_{i-1}, \psi) \\
&= \max_\pi \sum_{r \in \mathcal{R}} \mathbb{P}[r;\psi] \big(\max_{\pi' \in q_{i-1} \cup \{\pi\}} V^{\pi'}_r - \max_{\pi' \in q_{i-1}} V^{\pi'}_r\big) \\
&= \max_\pi \sum_{r \in \mathcal{R}} \mathbb{P}[r;\psi] \max\big(V^{\pi}_r - \max_{\pi' \in q_{i-1}} V^{\pi'}_r, 0\big)
\end{aligned} \tag{6}$$

Eq. 6 is what we want to optimize at each iteration to find π_i. One step of the procedure is illustrated in Figure 1. In the i-th iteration, we have $q_{i-1} = \{\pi_1, \ldots, \pi_{i-1}\}$. We find a policy π_i that maximizes the dark shaded area in the rightmost figure, which is the weighted (by ψ) margin that the added policy outperforms the policies already in the query.

The $\max$ operator inside the objective function makes this equivalent to a mixed integer linear programming (MILP) problem. Based on Eq. 3, we use x to denote the state-action occupancy measure of π_i, and rewrite the optimization problem in Eq. 6 as follows.

$$\max_{x,\{y_r\},\{z_r\}} \sum_{r\in\mathcal{R}} \mathbb{P}[r;\psi]y_r \tag{7}$$

$$\text{s.t.} y_r \le \sum_{s,a}[x(s,a)r(s,a)] - \max_{\pi\in q_{t-1}} V_r^\pi$$

$$+ M(1-z_r), \forall r\in\mathcal{R} \tag{7a}$$

$$y_r \le 0 + Mz_r, \forall r\in\mathcal{R} \tag{7b}$$

$$\sum_{a'} x(s',a') = \sum_{s,a} x(s,a)T(s,a,s') + \delta(s',s_0), \forall s' \tag{7c}$$

$$x(s,a) \ge 0, \forall s,a \tag{7d}$$

$$z_r \in \{0,1\}, \forall r\in\mathcal{R} \tag{7e}$$

We rewrite the $\max$ function of Eq. 6 as two constraints (corresponding to lines 7a and 7b), where at any given time only one of them applies. M is an arbitrarily large positive number. We introduce binary variables z_r for each $r\in\mathcal{R}$ so that when $z_r = 0$, the first constraint (line 7a) is relaxed, since $y_r \le M$ holds anyway, and the second constraint (line 7b) is enforced. When $z_r = 1$, the first constraint is enforced and the second constraint is relaxed. The constraints in lines 7c-7e ensure that x is a valid state-action occupancy.

Analysis. The EUS of an approximately-optimal policy query from this greedy process enjoys the guaranteed performance $EUS(q_k,\psi) \ge \eta EUS(q^*,\psi) = \eta EPU(q^*,\psi)$, where $\eta = 1-(\frac{k-1}{k})^k$ and q^* is the optimal k-response policy query (Viappiani and Boutilier 2010). Building a policy query with k elements requires solving k MILP problems. The complexity is polynomial in the number of state and action pairs, but combinatorial in the number of reward candidates in the worst case. For problems with many reward candidates, other approximation methods like policy gradient methods could be tried in the future.

Now we consider how we can characterize the policies we found by solving the optimization problem in Eq. 7. At the i-th iteration, let $x, \{y_r\}, \{z_r\}$ be solutions. π_i is the policy that occupancy x represents. Then:

$$\begin{aligned}\pi_i &= \arg\max_\pi \sum_{r\in\mathcal{R}} \mathbb{P}[r;\psi]y_r \\ &= \arg\max_\pi \sum_{r\in\mathcal{R}} \mathbb{P}[r;\psi]z_r(V_r^\pi - \max_{\pi'\in q_{t-1}} V_r^{\pi'}) \\ &= \arg\max_\pi \sum_{r\in\mathcal{R}} \mathbb{P}[r;\psi]z_r V_r^\pi\end{aligned}$$

That is, π_i maximizes the mean reward $\sum_{r\in\mathcal{R}}\mathbb{P}[r;\psi]z_r r$, which is the mean reward of a subset of the reward candidates. Note that this is not a way to find π_i, since $\{z_r\}$ are outputs of the optimization problem. The MILP problem decides the mean reward of which subset of reward candidates to optimize, leading to the following theorem.

Theorem 2. *Any policy in the optimal recommendation set must be the optimal policy under the mean reward of a subset of reward candidates.*

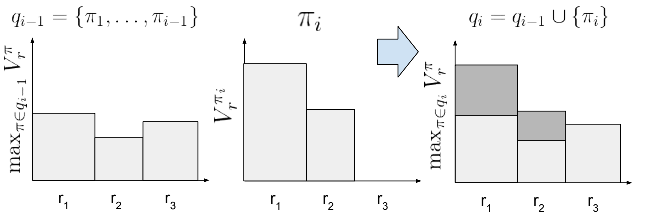

$q_{i-1} = \{\pi_1,\dots,\pi_{i-1}\}$
$\max_{\pi\in q_{i-1}} V_r^\pi$
π_i
$V_r^{\pi_i}$
$q_i = q_{i-1}\cup\{\pi_i\}$
$\max_{\pi\in q_i} V_r^\pi$
r_1
r_2
r_3

Figure 1: Greedy policy construction illustration.

Proof Sketch. If a policy in the optimal recommendation set is not an optimal policy under the mean reward of any subset of reward candidates, we can always find a subset of reward candidates $R\subseteq\mathcal{R}$ and replace this policy with the optimal policy of the mean reward of R. Then, the EUS of this policy query can only monotonically increase. □

Thus, to construct the approximately optimal k-response policy query, iteratively apply the MILP (Eq. 7) k times to generate k subsets of the reward candidates, and then find the optimal policy for the mean reward function of each subset. The query is composed of these k policies, and, importantly, the process avoids searching over all policies.

Projecting to Trajectory Queries

While the preceding is an effective approach for constructing approximately optimal policy queries, the size and complexity of policies can make queries about them hard for an operator to understand and answer. HRI settings thus often draw from a simpler set of *askable* queries, which we refer to as Q. It is impractical to compute the EPU of all of the queries in Q and select the best one when $|Q|$ is large. The strategy we advocate for more efficiently finding a good askable query from Q is to exploit our method for finding an approximately-optimal but possibly unaskable policy query. We do this using query projection (Cohn 2016). With q_Π^* being the approximately-optimal policy query found by the greedy construction method, and Q the set of askable queries, we find the query $q\in Q$ to ask as follows (Cohn 2016):

$$q = \arg\min_{q\in Q} H(q_\Pi^*|q) \tag{8}$$

where $H(q'|q)$ is the conditional entropy, defined as $H(q'|q) = \sum_{j\in q}\mathbb{P}[q \rightsquigarrow j;\psi]H(q'|q \rightsquigarrow j) = -\sum_{j\in q}\mathbb{P}[q\rightsquigarrow j;\psi]\sum_{j'\in q'}\mathbb{P}[q'\rightsquigarrow j'|q\rightsquigarrow j]\log\mathbb{P}[q'\rightsquigarrow j'|q\rightsquigarrow j]$. This approach uses the approximately-optimal policy query as a target for finding a good askable query. Concretely, we want the askable query that minimizes the robot's uncertainty about which response would have been chosen if the desired policy query could have been asked and answered. While the query found this way is not necessarily the optimal query in Q (because there might be no askable query that is a good proxy for asking q_Π^*), it often does well (as is shown empirically later) and avoids extensive computation.

We apply this projection technique in a setting where an askable query from the robot consists of a set of trajectories

(Wilson, Fern, and Tadepalli 2012) where, to ease comparison by the operator and to stay consistent with the literature, all of the trajectories begin from the same state and extend to the same finite horizon. Formally, a trajectory, $u = \{(s_1, a_1), \ldots, (s_l, a_l)\}$, is a sequence of state action pairs, where s_{t+1} is reached after taking a_t in s_t, and l is the length of each trajectory. A trajectory query is a set of trajectories, $q_U = \{u_1, \ldots, u_k\}$.

Using the query projection method described in Eq. 8, the robot finds the start state for the trajectory query based on q^*_Π as the state maximizing the heuristic value:

$$h^{QP}(s) = \mathbb{E}_{q_U \sim U(s)} \sum_{u \in q_U} \mathbb{P}(q_U \rightsquigarrow u; \psi) H(q^*_\Pi | q_U \rightsquigarrow u; \psi) \tag{9}$$

where $U(s)$ is a set of policies constructed in the following way. For each $\pi \in q^*_\Pi$, the robot identifies the set of reward candidates that π performs better on than other $\pi' \in q^*_\Pi$, and then one reward candidate is chosen from this set. The robot adds the optimal policy of this reward candidate to $U(s)$. q_U is a set of l-length prefixes of the trajectories starting from s drawn from the policies in $U(s)$. Note that a policy could have many different trajectories if the transition function is stochastic. Similar to (Akrour, Schoenauer, and Sebag 2013), the robot chooses a state that will generate good trajectory queries *in expectation*. The trajectory query that the robot actually asks is a set of trajectories, each of which is generated by following one policy in $U(s)$.

There are certainly other query projection methods in trajectory queries besides our straightforward choice. Note we did not use the greedily constructed policies themselves as $U(s)$ to generate trajectories. Those policies optimize over mean rewards of subsets of reward candidates, and thus may look very different from the optimal policy of any reward candidate.

Also, instead of first greedily forming policy queries and then projecting them to the trajectory query space, one could ask why we don't just greedily generate trajectory queries directly, which is what the Expected Belief Change and Query by Disagreement strategies of others (and that we soon describe) do. The reasons are two-fold. First, since trajectory queries are not decision queries, the greedy construction procedure applied to trajectories does not enjoy the near-optimality guarantee for policy queries, and thus could be arbitrarily suboptimal. We instead project from the k-ary policy query that the robot provably would want to ask (if it could) to find the k-ary trajectory query that best matches it. The second reason is that our projection technique generalizes to any spaces of"askable queries," while heuristics for greedily constructing trajectory queries don't necessarily generalize.

Empirical Evaluation

In the preceding, we have claimed the following: (1) Greedily-constructed policy queries can approximate optimal policy queries in performance while being computed more cheaply; and (2) Good trajectory queries can be found through projection of greedily-constructed policy queries. We now empirically test these claims.

Our evaluation assumes that the operator will respond to a query offering k choices with her most-preferred response, but we need to be more precise about exactly how that response is chosen. Given k policies (or trajectories) to choose from, there might be none that corresponds to what the operator would choose (particularly when the number of reward candidates n is larger than k). For policy queries, we have defined the response model to be that the operator responds with the policy that has the largest value under her reward function. We could do something similar for l-length trajectories, but the accumulated rewards up to a finite horizon may not well reflect the operator's preference (the operator may want to follow a trajectory to collect a distant reward). So, like (Wilson, Fern, and Tadepalli 2012), we assume the operator responds with the choice that is most *similar* to what she would do. To determine similarity, we assume a distance metric between two states s_1 and s_2, denoted by $d(s_1, s_2)$. This metric also applies to two trajectories u_1, u_2 of equal length: $d(u_1, u_2) = \sum_{t=1}^{l} d(u_{1,t}, u_{2,t})$, where $u_{i,t}$ is the state reached at the t-th time step in trajectory u_i. With this metric, the response model of a trajectory query q_U can be defined as follows: $q_U \rightsquigarrow u \implies \mathbb{E}_{u^* \sim \pi^*} d(u, u^*) \leq \mathbb{E}_{u^* \sim \pi^*} d(u', u^*), \forall u' \in q_U$. In this paper, we assume that, when computing the posterior distribution over reward candidates based on the operator's response (Eq. 4), the robot knows how the operator selects a response. Confusions that could arise if the robot incorrectly models the operator are left for future work.

Greedy Policy Queries

We compare the performance of our greedy construction method for policy queries against optimal policy queries in a version of the well-known Rock Sample domain (Smith and Simmons 2004), which we call rock collection, which we have purposely made to be sufficiently small for us to be able to exhaustively compute the optimal policy query. The robot navigates in a 5×30 gridworld, where it starts in the middle of the southern border and its actions are constrained to moving straight north, diagonally north-east, or diagonally northwest. The transition function is noisy: with probability 0.05, the robot reaches one of its north, north-east and north-west states uniformly randomly regardless of the action it takes. When the robot hits the boundary on either side, it will only move north rather than off the domain. The length of the trajectories it can follow is 4. There are m rocks randomly distributed in the grid, and the robot knows that the operator's true reward function is one of n equiprobable reward function candidates. We use the Manhattan distance as the distance metric between two states: $d((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$.

To generate the reward function candidates, we partition the m rocks into n subsets, and each reward candidate corresponds to a different partition containing rocks that the operator wants collected (positive reward), while all of the other rocks have rewards of -1. We consider three settings for the positive rewards, illustrated (using smaller grids) in Figure 2. **Reward setting #1:** The positive rewards of all the reward candidates are 1. **Reward setting #2:** The posi-

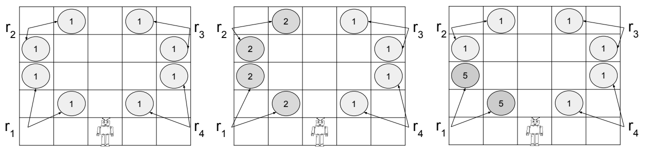


Figure 2: Reward settings of the rock collection domain. The locations of rocks are randomly assigned.

tive rewards of half of the reward candidates are 2, and the others are 1. **Reward setting #3:** The positive rewards of one reward candidate is 5, and the others are 1. An optimal query distinguishes the reward candidate with reward 5 from the other reward candidates.

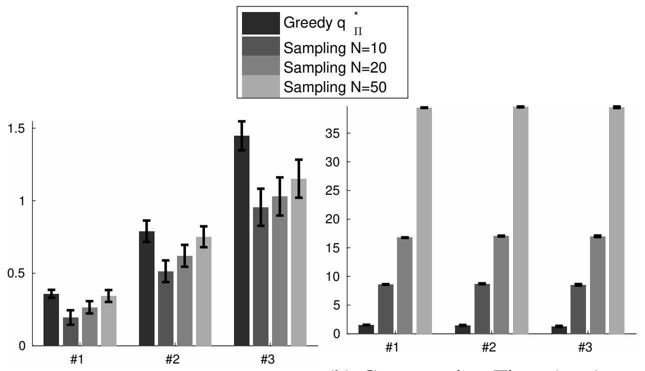
To confirm that the greedy construction method for policy querying can be effective, we compare it to optimal policy querying both in terms of EPU and computation time for $k = 2$. As we have discussed, searching in the policy space to find the optimal policy query is computationally intractable. We found in Theorem 2 that the optimal policy query maximizes the different subsets of the reward candidates. So, we enumerate all possible k subsets of reward candidates to find it. Note that although this avoids searching the policy space directly, it is only feasible for domains with a small number of reward candidates.

A natural question to ask is whether we really need to solve the MILP problem to find a policy query close to the optimal one. Can we randomly sample some policy queries and pick the best one among them? We consider a sampling approach that is similar to APRIL (Akrour, Schoenauer, and Sebag 2012). Inspired by Theorem 2, we randomly sample optimal policies of some mean reward $\bar{R}$, where $R \subseteq \mathcal{R}$, and then pick the set of policies that has the largest EUS. This avoids solving the MILP problem. We will refer to this method as "Sampling" in the following figure, and give the number of random samples N it considered.

Figure 3 shows that a greedy policy query is considerably cheaper to find, computationally, than any of the other approaches. Its $EVOI$ is comparable to the other approaches, always outperforming Sampling for $N = 10$, and even outperforming Sampling when $N = 50$ (and matching the optimal query) in reward setting #3 where the odds of randomly generating a policy that collects the few "good" rocks is small. This performance gap is caused by the fact that our approach finds a policy query by solving the optimization problem, rather than by sampling policy queries and hoping to find a good one.

Projecting to Trajectory Queries

Having established the existence of problems for which greedy policy query construction is warranted, we now turn to the question of whether projecting such queries into the trajectory space is effective. To answer this question, we compare our projection technique to two heuristic techniques from the literature. Both methods that we introduce below assume binary response queries and a general reward belief distribution where the optimal policies can be sampled from. We adapt their methods to our problem as follows.



(a) $EVOI$ vs. Reward Settings. (b) Computation Time (sec.) vs. Reward Settings.

Figure 3: Comparison of methods for finding policy queries in the rock collection domain. $n = 10, k = 2$.

Expected Belief Change is the heuristic with the best performance in (Wilson, Fern, and Tadepalli 2012). In our notation, the state it chooses for starting trajectories maximizes the following heuristic function.

$$h^{BC}(s) = \mathbb{E}_{q_U \sim U(s)} \mathbb{E}_{\psi' \sim \Psi(\psi, q_U)} \sum_i |\psi'_i - \psi_i| \quad (10)$$

where $U(s)$ is a set of optimal policies of k reward candidates starting from s, which are sampled according to ψ. As in the projection method, q_U is a set of l-length prefixes of trajectories starting from s drawn from the policies in $U(s)$. $\Psi(\psi, q_U)$ is the set of posterior beliefs by asking q_U with the prior belief ψ. This heuristic function estimates how much the belief distribution changes.

Query by Disagreement is another method proposed in (Wilson, Fern, and Tadepalli 2012), and the heuristic can be expressed in our notation as follows.

$$h^D(s) = \sum_{\pi, \pi' \in U(s)} \mathbb{E}_{u \sim \pi, u' \sim \pi'} d(u, u') \quad (11)$$

where $U(s)$ is a set of optimal policies of k reward candidates starting from s, which are sampled according to ψ. This heuristic function maximizes the expected distance between trajectories generated by the optimal policies of different reward candidates.

Using the same rock collection experimental setup as before, we compare the following algorithms.

1. **Optimal** q^*_Π is the optimal policy query.
2. **Greedy** q^*_Π is the policy query found by our greedy construction algorithm.
3. **Query Projection** finds a trajectory query using h^{QP} in Eq. 9.
4. **Belief Change** finds a trajectory query using h^{BC} in Eq. 10.
5. **Disagreement** finds a trajectory query using h^D in Eq. 11.
6. **Random Query** uniformly randomly picks a state and generates random trajectories from it.

Finding the optimal trajectory query is infeasible even for a small domain, so we use the optimal policy query as an upper bound.

The rock collection problem lets us adjust the number of reward candidates n and query choices k, so we can learn more deeply about the strengths and limitations of our projection approach. We report the results in Figure 5. Although we consider EPU as the objective so far, we evaluate our performance by $EVOI$, which is always nonnegative and best to measure the value of the query. The left column shows the case of 5 reward candidates ($n = 5$), and the right column has 10 ($n = 10$). From top to bottom, we increase the number of responses, k. The locations of rocks are randomly generated and different in different trials. 40 trials are run for each data point. To estimate the heuristic value for one state, we sample trajectories for 20 times to approximate the expectation ($\mathbb{E}_{q_U}$) in the heuristic functions. The error bars represent 95% confidence intervals. The variance of the results is caused by sampling trajectories from policies and the different initial configurations of the domain.

The cases where the number of reward candidates is 5 ($n = 5$) reinforce that greedy construction can perform well since the gap between the optimal policy query and the greedily constructed policy query is small. Once again, though, the time needed to find the optimal policy query is much longer (Figure 6). In fact, we can only find the optimal policy for the cases of $n = 5$ in this domain.

We next consider the cases with fewer responses (the top row, $n = 5, k = 2$ and $n = 10, k = 2$). We see that the query projection method outperforms the other heuristic methods. The reason is that our approach is aware of what the robot should ask given how few choices should be offered. Furthermore, the performance gap grows for reward settings #2 and #3 because the query projection method is aware of which reward candidates to distinguish from the others, while the other heuristic methods focus on reducing uncertainty of reward beliefs more broadly.

The advantage our query projection method has over the other methods decreases as the number of responses k grows relative to the number of reward candidates n: it is easier to elicit the operator's true reward function by asking queries with more responses, so being discriminating about which responses to offer matters less. We also observe the performance gap between the greedily constructed policy query and the rest grows because trajectory queries are less expressive than policy queries. For example, when $n = 10, k = 8$, it is difficult to find 8 trajectories starting from the same state that are as informative as an 8-policy query.

Now we compare the computation time between these methods in Figure 6. Clearly, optimal q^*_Π has the longest running time, and we can only afford computing it for $n = 5$. Greedy q^*_Π is much faster. Query projection is slightly slower than Greedy q^*_Π, since it needs to project the policy query it finds to the trajectory query space. Belief Change and Disagreement only need to compute a heuristic function for all states, which are both faster than query projection.

In summary, our experiments confirm that there exist domains where projecting into trajectory-query space can outperform previous strategies for trajectory query selection. Furthermore, they suggest that the projection technique's advantages are most pronounced when the number of query responses k is small relative to the number of reward candidates n, and when the space of trajectories is rich enough to highlight distinctions between policies for different reward candidate partitions. While much more efficient than searching in the trajectory query space directly, query projection runs longer than the previous methods. In some domains, a few more seconds of deliberation by the robot could be a worthwhile tradeoff for asking the operator a simpler (smaller k) and more enlightening query.

Multiple queries. As mentioned earlier, another performance metric is how the policy improves with more queries asked: the robot should elicit the human's true preference with the fewest queries possible (Akrour, Schoenauer, and Sebag 2012; Zhang and Stone 2015). We have instead assumed the robot asks only one query and should use it to elicit the most valuable information about the true preferences. If multiple queries can be asked, our projection technique can be applied iteratively, and we have empirically seen that when a few queries can be asked our technique still works best. However, as the number of queries grows, other approaches can be better at exploiting iterative querying. For example, if the robot can ask multiple binary response queries sequentially, then as the number of the queries approaches $\log_2 |\mathcal{R}|$, a good strategy is to select a query that rules out about half of the remaining reward candidates. Empirically, we have seen that the Expected Belief Change heuristic approximates such binary search. And as the number of queries nears the number of reward candidates, all the non-random approaches converge on the true preferences.

Discrete driving domain. The previous experiments were in a parameterized domain that we could flexibly modify to test our method under different conditions; we now briefly consider our method's performance in a less manipulable domain. The discrete driving domain we use is based on the driving domain of (Abbeel and Ng 2004). We consider five reward candidates. The human operator can be nice (avoid hitting other cars), nasty (prefer hitting other cars), dangerous (changing lanes right before hitting other cars), nice but prefer to stay in the left-most lane, or nice in the right-most lane. If the human is a nice driver, there is a large punishment (negative reward) for hitting another vehicle. All of these are from (Abbeel and Ng 2004), except the "dangerous driver" whom we added to increase the variety of the behaviors.

More details of our implementation are omitted, but are left out here because the main takeaway is that we, reassuringly, see similar results as before (Fig. 7). The query projection method has the same performance as the greedily constructed policy query. The Expected Belief Change heuristic has worse performance when $k = 2$, but it catches up when we increase k. Like Reward Setting #3 in the rock collection problem, distinguishing different reward candidates may yield significantly different EPU than heuristics that focus on reducing overall reward uncertainty. For example, we want to build a binary response query by considering

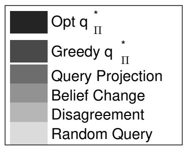
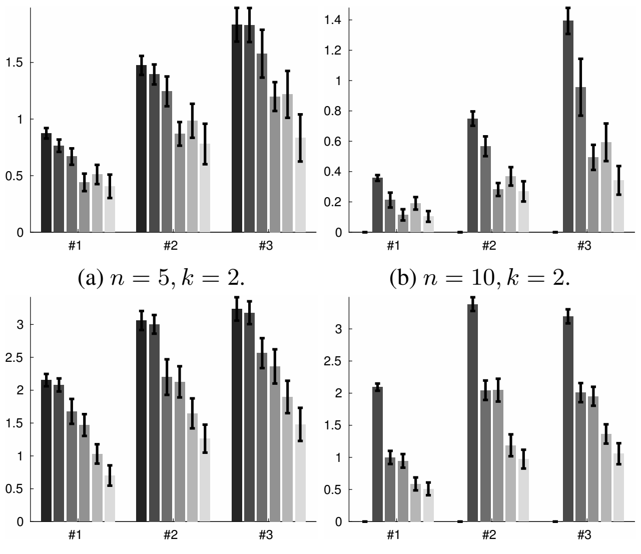


Figure 4: Legend for Figures 5, 6 and 7.



(a) $n = 5, k = 2$. (b) $n = 10, k = 2$.

(c) $n = 5, k = 4$. (d) $n = 10, k = 8$.

$EVOI$ vs. Reward Settings

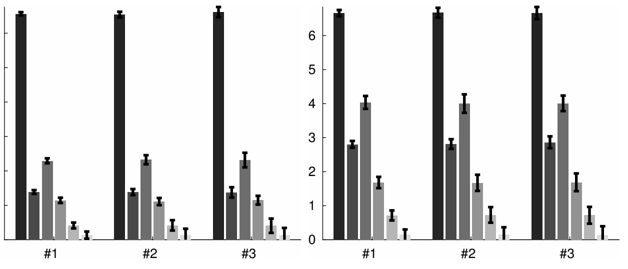
Figure 5: Comparison in the rock collection domain. Note that Opt q^*_Π in the settings with $n = 10$ cannot be computed so only placeholders are plotted.

the trajectories in the illustration in Figure 7. A good query would select the middle query and either one on the side to decide whether the human wants to hit another car. However, the competing methods do not discern such distinctions. Query by Disagreement selects the left-most and the right-most trajectory, and Expected Belief Change does not have a strong bias over these trajectories since the posterior beliefs are all changed.

Related Querying Work

The APRIL algorithm (Akrour, Schoenauer, and Sebag 2012) also looks at trajectory queries, but uses an iterative querying process where, instead of presenting k trajectories at once, it presents one trajectory at a time and essentially asks the operator a binary ($k = 2$) query: whether the new trajectory is preferred to the best trajectory shown so far. Furthermore, in that work the length of the trajectory is the full task horizon, rather than a shorter, specifiable horizon length l used in this paper. Subsequent work by those researchers considered the case where the operator might answer mistakenly (Akrour, Schoenauer, and Sebag 2013), which was not addressed in this paper.

Other querying methods are geared towards other types of operator models. If the operator has a specific rewarding goal state, for example, the robot might query to directly elicit the goal rather than rewards of intermediate states (Zhang and Stone 2015). Or the nature of a query response might differ, such as when the operator can answer by manipulating the robot to demonstrate a desired trajectory, rather than selecting from among offered trajectories (Jain et al. 2013). Demonstration from the operator can also improve aspects of the robot's model of the operator besides rewards, such as eliciting how to reach states, rather than preferences over states (Subramanian, Isbell Jr, and Thomaz 2016; Mohan and Laird 2014).



(a) $n = 5, k = 2$. (b) $n = 5, k = 4$.

Computation Time (sec.) vs. Reward Settings

Figure 6: Computational time of trajectory query selection methods in the rock collection domain.

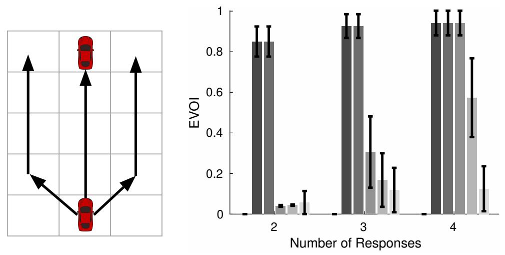


Figure 7: The driving domain and the evaluation. $n = 5, k = 2$. Opt q^*_Π cannot be computed.

Conclusion

When planning actions to please its operator, a robot may be uncertain about its operator's preferences. The robot should ask the query that most improves its plan. We have formulated a method for greedily constructing an approximately-optimal policy query that avoids enumerating all policy choices, and empirically shown that the approach can be particularly effective when randomly finding an informative policy option to offer is unlikely. We also presented an approach for projecting a policy query into a more easily askable space of trajectory queries, and showed empirically that the query found can be better than other trajectory-query selection strategies, particularly when the query should limit the trajectory choices to a small number. Future directions include introducing additional approximation techniques into the greedy construction method, finding approximately-optimal *sequences* of queries, and investigating the impact of a noisy operator response model.

Acknowledgements: Rob Cohn shared insights and technical details from his dissertation research on query projection that benefitted this paper, as did the comments from the anonymous reviewers. This work was supported in part by the Air Force Office of Scientific Research under grant FA9550-15-1-0039. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine learning*, 1–8.

Akrour, R.; Schoenauer, M.; and Sebag, M. 2012. APRIL: active preference learning-based reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 116–131.

Akrour, R.; Schoenauer, M.; and Sebag, M. 2013. Interactive robot education. In *ECML/PKDD Workshop on Reinforcement Learning with Generalized Feedback: Beyond Numeric Rewards*.

Cohn, R.; Durfee, E.; and Singh, S. 2011. Comparing action-query strategies in semi-autonomous agents. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 1102–1107.

Cohn, R.; Singh, S. P.; and Durfee, E. H. 2014. Characterizing EVOI-Sufficient k-Response Query Sets in Decision Problems. In *AISTATS*, 131–139.

Cohn, R. 2016. *Maximizing Expected Value of Information in Decision Problems by Querying on a Wish-to-Know Basis*. Ph.D. Dissertation, University of Michigan.

de Farias, D. P., and Van Roy, B. 2003. The linear programming approach to approximate dynamic programming. *Operations Research* 51(6):850–865.

Jain, A.; Wojcik, B.; Joachims, T.; and Saxena, A. 2013. Learning trajectory preferences for manipulators via iterative improvement. In *Advances in Neural Information Processing Systems*, 575–583.

Knox, W. B., and Stone, P. 2009. Interactively Shaping Agents via Human Reinforcement: The TAMER Framework. In *Proceedings of the Fifth International Conference on Knowledge Capture*, K-CAP '09, 9–16.

Mohan, S., and Laird, J. E. 2014. Learning goal-oriented hierarchical tasks from situated interactive instruction. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 387–394.

Ramachandran, D., and Amir, E. 2007. Bayesian inverse reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2586–2591.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 520–527.

Subramanian, K.; Isbell Jr, C. L.; and Thomaz, A. L. 2016. Exploration from demonstration for interactive reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 447–456.

Torrey, L., and Taylor, M. 2013. Teaching on a budget: Agents advising agents in reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, 1053–1060.

Viappiani, P., and Boutilier, C. 2010. Optimal Bayesian recommendation sets and myopically optimal choice query sets. In *Advances in Neural Information Processing Systems*, 2352–2360.

Wilson, A.; Fern, A.; and Tadepalli, P. 2012. A Bayesian approach for policy learning from trajectory preference queries. In *Advances in Neural Information Processing Systems*, 1133–1141.

Zhang, S., and Stone, P. 2015. CORPP: commonsense reasoning and probabilistic planning, as applied To dialog with a mobile robot. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 1394–1400.