Querying to Find a Safe Policy under Uncertain Safety Constraints in Markov Decision Processes

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

Abstract

An autonomous agent acting on behalf of a human user has the potential of causing side-effects that surprise the user in unsafe ways. When the agent cannot formulate a policy with only side-effects it knows are safe, it needs to selectively query the user about whether other useful side-effects are safe. Our goal is an algorithm that queries about as few potential side-effects as possible to find a safe policy, or to prove that none exists. We extend prior work on irreducible infeasible sets to also handle our problem’s complication that a constraint to avoid a side-effect cannot be relaxed without user permission. By proving that our objectives are also adaptive submodular, we devise a querying algorithm that we empirically show finds nearly-optimal queries with much less computation than a guaranteed-optimal approach, and outperforms competing approximate approaches.

Introduction

We consider a setting where an autonomous agent (called the “robot”) takes actions in a domain on behalf of a human user (called the “human”). The human specifies a goal or a reward function for the robot to optimize. However, in practice, she might overlook specifying some preferences, or purposely leave out details she (possibly incorrectly) thinks are irrelevant or already known by the robot.

For example, Figure 1 shows a robot navigation domain modified from our prior work (Zhang, Durfee, and Singh 2018). The human asks the robot to turn off a switch in the top-right corner. After planning, the robot could follow several possible policies. It could traverse the carpet (π₁), but get the carpet dirty. It could enter the room through door d₁ and then move away box b₁ or b₂, open door d₂, and go to the switch (π₂ or π₃ respectively). However, the robot is only told to change (turn off) the switch. Dirtying a carpet, moving boxes, or opening a door may be changes that negatively surprise the human, and thus executing any of these policies is potentially not safe. That is, because the robot might be new to this building, it may be uncertain about the safety constraints and human preferences here.

We can see that in this example, without communicating with the human, the robot could not find a safe policy since all the paths that reach the switch side effect some other objects. So we allow the robot to ask clarifying queries (for example, “Can I move box b₁ away?”). The robot finds a policy that is known to be safe when it queries and is explicitly told that the carpet can be dirty, or b₁ is movable and d₂ can be opened, etc. Our main focus is on how the robot should ask the minimum number of queries in expectation to either find a safe policy or prove that no safe policy exists.

The literature has explored the problem of avoiding negative side-effects (Amodei et al. 2016; Leike et al. 2017; Turner, Hadfield-Menell, and Tadepalli 2019). Our prior work (Zhang, Durfee, and Singh 2018) considered a similar problem, with the major difference being that we there assumed the robot initially knows a safe policy (to reach the switch without side effecting any objects other than ones it knows it can) and queries to find a maximally better safe policy. In our problem here, since no initial safe policy is known, our prior algorithm cannot be applied. This paper’s contribution thus is to answer the question of how the robot should query to find a safe policy or prove that no safe policies actually exist when initially no safe policies are known. Essentially, our problem is to query to satisfy safety constraints rather than to optimize a policy under safety constraints, which requires a very different approach despite the setting being similar. Note that once the robot has found a safe policy, if it can still ask more queries then it can use the prior algorithms to pose queries to improve the policy.

In this paper, we make the following contributions: 1) We formulate the problem of finding an initial safe policy in a factored Markov decision process. 2) We design a query algorithm that makes novel use of prior work on irreducible infeasible sets and adaptive submodularity. 3) Empirically, we
show that our algorithm finds nearly-optimal queries with much less computation than a guaranteed-optimal approach, and outperforms some computationally cheaper alternatives.

**Problem Statement**

Similar to our prior work (Zhang, Durfee, and Singh 2018) (hereafter referred to as ZDS in this paper), we model the domain as a factored Markov decision process (Boutilier, Dean, and Hanks 1999). A finite Markov decision process (Sutton and Barto 2018) is a tuple $(S, A, T, r, s_0, \gamma)$, with finite state space $S$, finite action set $A$, and transition function $T$ where $T(s, a, s')$ is the probability of reaching state $s'$ by taking action $a$ in $s$. $r(s, a)$ is the reward of taking action $a$ in $s$. $s_0$ is the initial state and $\gamma$ is the discount factor. In a factored MDP, a state is described in terms of values of various features (e.g., the robot’s location, cleanliness of each carpet, etc.), so the state space $S$ is the cross-product of the values the features can take. We will consistently use $\phi$ to denote one feature and $\Phi$ to denote a set of features.

We assume that the robot can partition the features into the following sets, exactly as we did in ZDS:

- $\Phi^F$: The (known-to-be-)free features. The robot knows these features are freely changeable (e.g., its location).
- $\Phi^L$: The (known-to-be-)locked features. The robot knows it should never change any of these features.
- $\Phi^R$: The unknown features. These are features that the robot does not know whether the user considers freely changeable or locked.

The human partitions features into two sets, $\Phi^F_L$ and $\Phi^H_F$. Per ZDS, the robot’s knowledge is consistent with that of the human, that is, $\Phi^R_F \subseteq \Phi^L^F$ and $\Phi^R_L \subseteq \Phi^H_L$. In general, the robot may not know the categories of all features ($\Phi^R_\Phi \neq \emptyset$).

As in our work in ZDS, to guarantee safety, the robot should never change features in $\Phi^H_F_L$. However, the robot only has partial information about this set. So, like in ZDS, we require that the robot should not change any features in $\Phi^F_L$ or $\Phi^H_F$ (Requirement 1). That is, it can only change features that are known to be free (in $\Phi^F_F$).

Only requiring that the robot not change these features may not lead to satisfactory policies. For example, in Figure 1, the robot can simply stay in place to avoid changing any unknown features. Though safe, this behavior receives no positive rewards, and we do not want the robot to follow such a trivially safe policy. Hence, we require that the robot occupy some goal states, denoted by $S_G$: The robot must eventually occupy goal states ($S_G$) with occupancy at least $\delta_G$ (Requirement 2). The robot knows both $S_G$ and $\delta_G$. We will define occupancy later in Eq. 1.

We refer to policies satisfying both requirements above as safe policies, in that they do not negatively surprise the human with unwanted side-effects or by not achieving the goal. Initially, the robot may not have a safe policy, as in Figure 1 where the carpet, boxes, and doors are unknown features. We allow the robot to pose iterative queries, following the convention in the literature (Regan and Boutilier 2009; Akourr, Schoenauer, and Sebag 2012; Le et al. 2018). Similar to feature queries in Cakmak and Thomaz (2012), the robot can query about an unknown feature $\phi \in \Phi^R$, then receive the human’s response ($\phi \in \Phi^H_F$ or $\phi \in \Phi^H_L$), and move the queried feature from $\Phi^F$ to $\Phi^H_F$ or $\Phi^H_L$. As needed, it can then repeat this process. In this paper, we assume that the feature representation is given to the robot and understood by the human (in our example, the state of a carpet, the position of a box, etc.).

The robot stops querying when it reaches one of the following two possible outcomes. **Outcome 1** (denoted by $\top$): The robot is able to find one or more safe policies after querying. Then it would follow an optimal safe policy. This happens when it knows enough features are free (for example, in Figure 1, if carpet is free or $b1$ and $d2$ are free). **Outcome 2** (denoted by $\bot$): The robot determines that no safe policy exists. This happens when it knows enough features are locked (for example, in Figure 1, carpet and $d2$ are both locked). Then the robot should terminate and inform the human that it cannot achieve the goal safely.

Note that given enough queries, the robot would eventually reach one of the two outcomes. The worst case would be that it queries about all unknown features, in which case it recovers $\Phi^F_F$ and $\Phi^F_L$. However, to reduce the human’s cognitive load, the robot wants to ask about the fewest unknown features possible to reach either outcome. As is common in the literature (Ramachandran and Amir 2007; Mindermann et al. 2018), we assume that the robot has a prior over the human’s preferences. It knows the probability of any unknown feature $\phi \in \Phi^R$ being free, denoted by $p_F(\phi)$. The probability that $\phi$ is locked is $p_L(\phi) = 1 - p_F(\phi)$. The robot’s objective is to minimize the number of queries in expectation to either find a safe policy (reaching $\top$) or determine that no safe policy exists (reaching $\bot$).

In this paper, we explain how finding the exact optimal query policy (that maps a partition of features to the optimal feature to query about) is computationally intractable, while some straightforward heuristics that select myopically-greedy queries can perform poorly under some circumstances. Our contribution is to identify and exploit set-cover subproblems within our problem to design an algorithm that uses superior heuristics in a myopically-greedy way.

**Background**

In this section, we first explain how to find safely-optimal policies given a partition of features. Then we describe dominating policies and relevant features that we will use in our algorithm.

**Finding Safely-Optimal Policies**

Given a partition of features ($\Phi^R_F$, $\Phi^R_F$ and $\Phi^R_L$), the robot wants to decide if a safe policy exists, and if it does, the robot wants to find a safely-optimal policy. A safely-optimal policy is a safe policy (satisfying Requirements 1 and 2) that has the optimal value, depending on the partition of features. It is easier to use linear programming (de Farias and Van Roy 2003) than dynamic programming methods like value iteration to find a safely-optimal policy because we can easily add constraints to it. The following LP 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 \( s \).

\[
\max_s \sum_{s, a} x(s, a) r(s, a) \tag{1}
\]

\[
s.t. \sum_{a'} x(s', a') = \gamma \sum_{s, a} x(s, a) T(s, a, s') + 1_{[\pi' = \pi_0]}, \forall s' \in S
\sum_{s \in S_G, a \in A} x(s, a) \geq \delta_G
\sum_{s \in S_{\pi'}, a \in A} x(s, a) = 0, \phi \in \Phi^F \cup \Phi^R
\]

where \( S_{\pi'} = \{ s : \phi(s) \neq \phi(s_0) \} \) is the set of states whose value for feature \( \phi \) differs from the initial state. The first constraint is a flow-conservation constraint that encodes the transition function. The second constraint makes sure that the robot reaches \( S_G \) with occupancy at least \( \delta_G \). The third constraint prevents the robot from changing any known-to-be-locked or unknown features. The safely-optimal policy is \( \pi(s, a) = x(s, a)/\sum_{a' \in A} x(s, a') \) for states where \( \sum_{a' \in A} x(s, a') > 0 \) (de Farias and Van Roy 2003). Other states are not visited by the safely-optimal policy so the actions to take in those states are not defined. In this paper, we require the robot to guarantee to never change any known-to-be-locked or unknown features. If we want to instead upper bound the robot’s occupancy on unsafe states (for example, when the transitions are stochastic), we can change the third constraint accordingly.

**Dominating Policies and Relevant Features**

We use some concepts from ZDS to help understand which policies and features we potentially need to consider. First, we compute the set of policies that the robot may consider following under any partitions of features. We call these policies **dominating policies**, denoted by \( \Gamma \).

**Definition 1. (ZDS)** A **dominating policy** is a safely-optimal policy for the circumstance where for some \( \Phi \subseteq \Phi^F \), \( \Phi \) are locked features and \( \Phi^R \setminus \Phi \) are free features.

To find all dominating policies, we could enumerate all \( \Phi \subseteq \Phi^R \), find the safely-optimal policy when \( \Phi \cup \Phi^F \) are locked features and, if it exists, add it to the set of dominating policies. Instead we use an algorithm (DomPolicies) from ZDS that is more efficient than this brute-force way to find dominating policies.

In ZDS, we further refer to the unknown features changed by a dominating policy as **relevant features**:

**Definition 2. (ZDS)** The **relevant features** of policy \( \pi \) is the set of initially unknown features changed by \( \pi \), denoted \( \Phi_{rel}(\pi) \).

The set of all relevant features is the union of the relevant features of all dominating policies, \( \Phi_{rel} = \bigcup_{\pi \in \Gamma} \Phi_{rel}(\pi) \).

As we observed in ZDS, the robot only needs to consider querying about relevant features.

**Example 1.** In Figure 2, a robot is tasked to turn off a switch. It changes any feature it visits. Illustrated in (a),

\[
\begin{align*}
\text{(a)} & \quad \Phi_1 \quad \Phi_2 \quad \Phi_3 \\
\text{(b)} & \quad \Phi_1 \quad \Phi_2 \quad \Phi_3 \\
\text{(c)} & \quad \Phi_1 \quad \Phi_2 \quad \Phi_3 
\end{align*}
\]

there are two dominating policies. \( \pi_1 \) (the upper path) changes \( \phi_1, \phi_2; \pi_2 \) (the lower path) changes \( \phi_1, \phi_3 \). So \( \Phi_{rel}(\pi_1) = \{ \phi_1, \phi_2 \}; \Phi_{rel}(\pi_2) = \{ \phi_1, \phi_3 \} \).

\( \Phi_{rel} = \{ \phi_1, \phi_2, \phi_3 \} \). So \( \phi_4 \) is not a relevant feature, which means that knowing \( \phi_4 \) is free does not help us to find a safe policy, and knowing \( \phi_4 \) is locked does not help us to prove that no safe policy exists.

**Querying to Find a Safe Policy**

In this section, we consider algorithms for query selection. We first consider some candidate algorithms, including computing the optimal query policy and straightforward heuristics that myopically select queries. Then we describe our main contribution, a set-cover approach for query selection.

We use \( (\Phi^F_{\pi'}, \Phi^F_{\psi}, \Phi^F_{\psi}) \) to refer to the robot’s partition of features. When we want to be precise about the updated partition of features during the query process, we use \( \psi \) to denote the current partition of features: \( \psi = (\Phi^F_{\pi'}, \Phi^F_{\psi}, \Phi^F_{\psi}) \).

Let \( \psi_0 \) be the initial partition. Clearly, \( \Phi^F_{\pi'} \subseteq \Phi^F_{\pi'} \subseteq \Phi^F_{\pi'} \) for all \( \psi \) reached by any query policy. We denote by \( \pi_q \) a **query policy**. Given a partition \( \psi \), \( \pi_q(\psi) \) is an unknown feature that it queries about next, or \( \top \) or \( \bot \) when it knows that safe policies exist or not. A query policy can be illustrated as a decision tree (for example, Figure 2 (c)). Suppose we want to find the optimal query policy by considering all possible query policies (enumerating all possible decision trees as illustrated). The number of possible query policies is exponential in the number of relevant features. So we would prefer not to exhaustively evaluate all possible queries to find the optimal one.

**Myopic Query Selection**

Although finding the provably-optimal query policy is intractable, we can take advantage of the fact that we do not need to find a complete query policy before posing a query. We can instead myopically select a feature to query about, and then decide on the next feature (if any) to query about.
Algorithm 1 Myopic Query Selection

1: given query selection criterion $h$, initial partitions $\Phi_F^{\psi_0}, \Phi_L^{\psi_0}, \Phi_T^{\psi_0}$
2: $\psi \leftarrow \psi_0$
3: while not ($\top$ or $\bot$) do
4: choose $\phi_q$ according to $h$ based on partition $\psi$
5: query the human about $\phi_q$
6: $\Phi_F^{\psi'} \leftarrow \Phi_F^{\psi} \setminus \{\phi_q\}$
7: if $\phi_q$ is free then
8: $\Phi_F^{\psi'} \leftarrow \Phi_F^{\psi'} \cup \{\phi_q\}$
9: else
10: $\Phi_L^{\psi'} \leftarrow \Phi_L^{\psi'} \cup \{\phi_q\}$
11: end if
12: $\psi \leftarrow \psi'$
13: update $\top$ or $\bot$ based on the new partition
14: end while
15: if $\top$ is true then
16: return safely-optimal policy under $\psi$ by solving Eq. 1
17: else
18: return “No safe policy exists.”
19: end if

depending on the human’s response. Algorithm 1 gives the skeleton of this myopic querying procedure. The robot starts with its initial partitions of features and a query selection criterion $h$ as input. Until it reaches one outcome or the other, it selects a feature to query about using $h$, and updates its partition of features based on the human’s response.

We use Eq. 1 in Line 13 to update the truth values of $\top$ and $\bot$. If Eq. 1 has a feasible solution under the new partition of features, then the robot finds a safe policy. If Eq. 1 does not have a solution even if $\Phi_F^{\psi'}$ are all free features, then the robot knows that no safe policy exists.

One may easily come up with straightforward heuristics for $h$ used in Line 4. For example, the robot can focus on the policy that is most likely to be safe and query about its relevant features that are still unknown. Concretely, the robot first finds all dominating policies. Then it finds the policy that is most likely to be safe: $\text{arg max}_{\pi \in \Gamma} \prod_{\phi \in \Phi_{rel}(\pi)} p_F(\phi)$. Then it queries about that policy’s relevant (unknown) feature that has the largest value of $p_F(\cdot)$. We refer to this heuristic as most-likely-policy. We see in the following example that it does not always find the optimal query policy.

Example 2. In Figure 2, let $p_F(\phi_1) = 0.5$, $p_F(\phi_2) = 0.6$. Aiming to find a safe policy, the most-likely-policy heuristic would first query about $\phi_2$ or $\phi_3$ since its $p_F$ value is higher. However, it is easy to verify that an optimal query policy would start with querying about $\phi_1$ first. The optimal query policy, illustrated in Figure 2 (c), in expectation asks about 1.7 features, while using the most-likely-policy asks about 2.24.

We will describe other heuristics in the Empirical Evaluation section as possible candidates, but in the following subsections, we first introduce our main contribution: we reveal a set-cover structure in the problem and exploit the structural properties to select better queries.

Set Cover Formulation

In this subsection, we first show that reaching $\top$ and $\bot$ are each equivalent to a set cover problem. Non-existence of safe policies. It is easy to see that the robot finds a safe policy if the relevant features of a dominating policy are all known to be free. Formally, $\top \iff \exists \pi \in \Gamma, \Phi_F \supseteq \Phi_{rel}(\pi)$. This is not a set cover problem and does not give us insights on which feature to query about. But we can use this observation to make the following claim about when safe policies do not exist. The robot knows that no safe policy exists if the relevant features of all dominating policies contain at least one known-to-be-locked feature. Indeed, if for all dominating policies, we know some of their relevant features are locked, then it is impossible to find a safe policy. Formally, let $S_{rel} = \{\Phi_{rel}(\pi) : \pi \in \Gamma\}$. Then

$$\bot \iff \forall \Phi \in S_{rel}, \Phi_L^T \cap \Phi \neq \emptyset.$$ (2)

Example 3. In Figure 2 (a), $S_{rel} = \{\{\phi_1, \phi_2\}, \{\phi_1, \phi_3\}\}$. If the robot knows that $\phi_1$ is a locked feature, or $\phi_2$ and $\phi_3$ are both locked features, then a safe policy does not exist.

Existence of safe policies. Relevant features of dominating policies ($S_{rel}$) can help us determine non-existence of safe policies, but they do not directly help in finding a safe policy. We could focus on one safe policy that is most likely to be safe, similar to the simple most-likely-policy heuristic (Example 2). Instead, though, we now show that existence of safe policies can be mapped to a different set cover problem.

Our crucial insight is recognizing that our problem is similar to the maximum feasible set problem in linear programming (LP) (Chinneck 2007): When an LP problem is infeasible, the objective is to find the minimum number of constraints to remove to make the problem feasible. The analogy in our problem is that, when initially imposing the constraints of all unknown and locked features, the LP problem (Eq. 1) is infeasible. The key difference is that our objective is not to find the minimum number of constraints to remove since the robot cannot arbitrarily remove constraints. It can only decide which features to query about and remove the corresponding constraints when they are known to be free. Nonetheless, we can adapt tools in the literature for finding maximum feasible sets to our needs.

To identify maximum feasible sets, Chinneck (2007) introduced the concept of an irreducible infeasible set (IIS). We adopt the IIS concept in our context as follows.

Definition 3. A subset of unknown features $\Phi \subseteq \Phi_L^T$ is an irreducible infeasible set (IIS) if 1) safe policies do not exist if $\Phi$ are locked features and $\Phi_L^T \setminus \Phi$ are free features; 2) once any feature $\phi \in \Phi$ is known to be free (that is, $\Phi \setminus \{\phi\}$ are locked features and $\Phi_L^T \setminus (\Phi \setminus \{\phi\}$ are free features), the robot can find a safe policy.

We observe that the robot can find a safe policy if there exists at least one known-to-be-free feature in all IISs. Otherwise, if there are any IISs that only contain unknown or known-to-be-locked features, the robot has not yet found a
safe policy. Define $S_{IIS}$ to be the set of IISs induced by all relevant unknown features. Formally,
\[ \top \iff \forall \Phi \in S_{IIS}, \Phi^R \cap \Phi \neq \emptyset. \] (3)

**Example 4.** In Figure 2(b), $S_{IIS} = \{\{\phi_1\}, \{\phi_2, \phi_3\}\}$. If the robot knows that $\phi_1$ is a free feature, and at least one of $\phi_2$ or $\phi_3$ is a free feature, then a safe policy must exist.

To compute $S_{IIS}$ and $S_{rel}$, we use Algorithm DomPolicies from ZDS to first find dominating policies and their relevant features, $S_{rel}$ (with the computational complexity of $O(2^{\phi_{rel}})$ in the worst case). Then to compute $S_{IIS}$, we observe that in our problem, $S_{IIS} = \{\Phi \subseteq S_{rel} : |\Phi \cap \Phi' \cap \Phi^R \cap \Phi = \emptyset, \forall \Phi' \in S_{rel}\}$. By the definition of IIS, if $\Phi \in S_{IIS}$ are locked features (and $\Phi^R \setminus \Phi$ are free), no safe policy exists because every dominating policy has a relevant feature that is locked. If any $\phi \in \Phi$ is free, we can find a safe policy because for some dominating policies, the only presumed locked feature is now free.

**Set-Cover-Based Algorithm**

Our set-cover-based algorithm specializes Algorithm 1 by using query responses to not only update the partition of features, but also update $S_{IIS}$ and $S_{rel}$. Denote the two sets under partition $\psi$ by $S_{IIS}^\psi, S_{rel}^\psi$. Specifically, it initially computes $S_{IIS}^{\psi_0}$ and $S_{rel}^{\psi_0}$ under the initial partition $\psi_0$. It needs to maintain these two sets in the query process. Suppose the robot has partition $\psi$ and queries about feature $\phi_q$. If $\phi_q$ is free, then the sets in $S_{IIS}^{\psi}$ that contain $\phi_q$ are covered, so they are removed from $S_{IIS}^{\psi}$. Also, since $\phi_q$ is free, it cannot be used to cover sets in $S_{rel}^{\psi}$. We remove it from the sets in $S_{rel}^{\psi}$, meaning that the sets originally containing $\phi_q$ need to be covered by some other feature. Formally,
\[ S_{IIS}^{\psi'} \leftarrow S_{IIS}^{\psi} \setminus \{\Phi \in S_{IIS}^{\psi_0} : \phi_q \in \Phi\}; \] (4)
\[ S_{rel}^{\psi'} \leftarrow \{\Phi \setminus \{\phi_q\} : \Phi \in S_{rel}^{\psi}\}. \] (5)

Similarly, if $\phi_q$ is locked, we update both sets as follows,
\[ S_{IIS}^{\psi'} \leftarrow S_{IIS}^{\psi} \setminus \{\Phi \in S_{rel}^{\psi_0} : \phi_q \in \Phi\}; \] (6)
\[ S_{rel}^{\psi'} \leftarrow \{\Phi \setminus \{\phi_q\} : \Phi \in S_{IIS}^{\psi_0}\}. \] (7)

Finally, for the set-cover-based version of Algorithm 1, in Line 13, we can avoid the expense of solving Eq. 1, because $\top$ (or $\perp$) is true simply if $S_{IIS}^{\psi}$ (or $S_{rel}^{\psi}$) has become empty, meaning it is fully covered.

We can also use $S_{IIS}^{\psi}$ and $S_{rel}^{\psi}$ to compute better heuristics $h$ for Line 4. To describe how, we first show that our objectives are adaptive submodular.

**Adaptive submodularity.** While we have shown $\top$ and $\perp$ are each equivalent to a set cover problem, we should again note that they differ from classic set cover problems (Williamson and Shmoys 2011) because the robot cannot simply assign a feature to be free or locked (in Figure 2, it cannot assert $\phi_1$ is a locked feature so that it can cover $S_{rel}$ using just one query). It can only choose which feature to query about, and update the feature’s category based on the human’s response.

Let $f_{IIS}, f_{rel}$ be the current coverage under partition $\psi$:
\[ f_{IIS}(\psi) = |\{\Phi \in S_{IIS}^{\psi_0} : \Phi^R \cap \Phi = \emptyset\}| \] (8)
\[ f_{rel}(\psi) = |\{\Phi \in S_{rel}^{\psi_0} : \Phi^R \cap \Phi = \emptyset\}| \] (9)

We further define the one-step gain in coverage as follows, similar to $\Delta$ in Golovin and Krause (2011) (referred to as GK from here on): $\Delta_{IIS}(\phi|\psi) = p_F(\phi) \cdot |S_{IIS}^{\psi_0}[\phi]|$, where $S_{IIS}^{\psi_0}[\phi]$ is the number of sets in $S_{IIS}^\psi$ that contain $\phi$; $\Delta_{rel}(\phi|\psi) = p_L(\phi) \cdot |S_{rel}^{\psi_0}[\phi]|$, where $S_{rel}^{\psi_0}[\phi]$ is defined similarly. We now show that $f_{IIS}$ and $f_{rel}$ are adaptive submodular functions (GK) based on the subset relation: $\psi \subseteq \psi' \iff (\Phi^F_F \subseteq \Phi^F_F \wedge \Phi^F_L \subseteq \Phi^F_L)$.  

**Theorem 1.** $f_{IIS}$ and $f_{rel}$ are both adaptive submodular functions under $\subseteq$.

**Proof Sketch.** Take $f_{IIS}$ for example. It is sufficient to show that for any $\psi, \psi'$ where $\psi \subseteq \psi'$ and any feature $\phi$, $\Delta_{IIS}(\phi|\psi) \geq \Delta_{IIS}(\phi|\psi')$. We observe that $\Delta_{IIS}(\phi|\psi) = p_F(\phi) |S_{IIS}^{\psi_0}[\phi]|$ if $\phi \in \Phi^F_F$, and 0 otherwise. Since $\psi \subseteq \psi'$, $\Phi^F_F \supseteq \Phi^F_F$, that is, unknown features monotonically vanish over time. We can verify that $\Delta_{IIS}(\phi|\psi) \geq \Delta_{IIS}(\phi|\psi')$, which completes the proof. }

One benefit of this result is the following. GK implied that if the robot knows that it can cover $S_{IIS}$, it is approximately-optimal to choose $\arg \max_\phi \Delta_{IIS}(\phi|\psi)$. Similarly, if it knows that it can cover $S_{rel}$, the robot should choose $\arg \max_\phi \Delta_{rel}(\phi|\psi)$. However, in our problem, the robot does not know which outcome ($\top$ or $\perp$) is true before it finishes querying. So we need our algorithm to balance between two possible objectives.

The rest of this section describes two query-selection heuristics. The first (set-cover heuristic) is more straightforward and easier to compute. The second (inverse-cover-ratio heuristic) is comparatively more expensive to compute. We will empirically test to see if the second one finds a better query that is worth the extra computation in the Empirical Evaluation section.

**Set-cover heuristic** ($h_{SC}$). One simple way of combining two objectives is the sum of both, weighted by the inverse of the number of sets remaining to cover.
\[ h_{SC}(\phi; \psi) = \Delta_{IIS}(\phi|\psi) \cdot |S_{IIS}^{\psi_0}| + \Delta_{rel}(\phi|\psi) \cdot |S_{rel}^{\psi_0}| \] (10)

We can expect this heuristic to have approximately-optimal performance when $p_F$ for all unknown features approaches 0 or 1. In these cases, it would focus on covering $S_{IIS}$ (or $S_{rel}$) by maximizing $\Delta_{IIS}$ (or $\Delta_{rel}$), which GK show to be approximately-optimal. When the probabilities of changeability vary, we no longer have a pure set cover problem and it is difficult to provide a theoretical guarantee. The robot’s strategy is to query the feature that makes the most progress (in expectation) in both set cover problems at once. We see in the following example that $h_{SC}$ does find an optimal query policy in the example in Figure 2.
Example 5. We consider again the domain in Figure 2. \( p_F(\phi_1) = 0.5, p_F(\phi_2) = p_F(\phi_3) = 0.6 \). Consider the first query to pose, \( h_{SC}(\phi; \psi_0) = 0.5 \cdot 1/2 + 0.5 \cdot 2/2 = 0.75 \). \( h_{SC}(\phi_2; \psi_0) = h_{SC}(\phi_3; \psi_0) = 0.6 \cdot 1/2 + 0.4 \cdot 1/2 = 0.5 \). Higher heuristic values mean more coverage. So the algo-

Inverse-coverage-ratio heuristic (IICR). The heuristic we describe below uses some properties of \( I_{IIS} \) and \( I_{rel} \) (Eqs. 8 and 9). We first describe a theorem adapted from GK. Let \( c(\pi_q^*|\psi) \) be the expected number of features queried by an optimal query policy \( \pi_q^* \) starting from \( \psi \).

**Theorem 2.** (Adapted from Lemma A.9 in Golovin and Krause (2011)) Starting from partition \( \psi \), the optimal query policy \( \pi_q^* \) has the following property, 

\[
c(\pi_q^*|\psi) \geq \frac{\Delta_{IIS}(\pi_q^*|\psi)}{\max_\phi \Delta_{IIS}(\phi|\psi)}.
\]

where \( \Delta_{IIS}(\pi_q^*|\psi) \) is the number of IISs \( \pi_q^* \) can cover in ex-

We call the left-hand-side the **inverse coverage ratio** for \( S_{IIS} \). Intuitively, the inequality says the follow-

We define \( ICR_{IIS}(\psi) \) and \( ICR_{rel}(\psi) \) similarly, which has a similar property.

\[
ICR_{IIS}(\psi) := \frac{P[T; \psi] \cdot |S_{IIS}|}{\max_\phi \Delta_{IIS}(\phi|\psi)} \leq c(\pi_q^*|\psi). \tag{13}
\]

The inverse coverage ratio is defined as the sum of the two. 

\[
ICR(\psi) = ICR_{IIS}(\psi) + ICR_{rel}(\psi) \tag{15}
\]

Note that \( ICR \) is not necessarily a lower bound, but intuitu-

The states where either outcome (\( T \) or \( \bot \)) is reached have re-

\[
\phi_q = \arg \min \phi h_{ICR}(\phi; \psi) \tag{16}
\]

\[
h_{ICR}(\phi; \psi) = p_F(\phi) ICR(\psi|\psi \in \Phi_F^R) + p_L(\phi) ICR(\psi|\psi \in \Phi_L^R) \tag{17}
\]

where \( \psi|\phi \in \Phi_F^R \) is the partition of features that, starting from partition \( \psi \), the robot queries about \( \phi \) and confirms that it is a free feature. \( \psi|\phi \in \Phi_L^R \) is defined similarly. It consid-

In Figure 2, let \( p_F(\phi_1) = 0.9, p_F(\phi_2) = p_F(\phi_3) = 0.1 \). \( h_{SC}(\phi_1) = .9 \cdot 1/2 + .1 \cdot 2/2 = .55 \); \( h_{SC}(\phi_2) = h_{SC}(\phi_3) = .1 \cdot 1/2 + .9 \cdot 1/2 = .5 \). So \( h_{SC} \) would first query about \( \phi_1 \). However, since safe policies are unlikely to exist, by computing \( P[T; \psi] \) and \( P[\bot; \psi] \), \( h_{ICR} \) would query about \( \phi_2 \) and (if it’s locked) then \( \phi_3 \) aiming to prove that no safe policy exists, which asks fewer queries in ex-

**Empirical Evaluation**

Here we empirically answer the following three questions:

**Q1:** Are queries found by \( h_{SC} \) or \( h_{ICR} \) close to the optimal query, while computationally much cheaper?

**Q2:** Are queries found by \( h_{SC} \) or \( h_{ICR} \) better than queries based on simpler and cheaper heuristics?

**Q3:** Is the additional cost of \( h_{ICR} \) over \( h_{SC} \) worth it?

We consider a variation of our earlier (ZDS) robot navigation domain for evaluation, illustrated in Figure 3 (left). The size of the domain is \( 6 \times 6 \), with 5 randomly-generated walls which the robot cannot move through so we have vari-

The states where the switch is off; \( \gamma = 0.9 \). There are carpets randomly placed in this environ-

When the robot traverses a carpet, it makes the carpet dirty. For each carpet, whether the robot is allowed to make it dirty is unknown to the robot. So each carpet corresponds to an unknown feature. All carpets are initially clean and the number of carpets varies in different experiments.

We compare several strategies and heuristics for query sele-

One, **optimal**, finds the exact optimal query policy: Consider a Query-MDP, \( M^Q \), where states are partitions of features. Actions are queries that probabilistically transit be-

The states where the switch is on; \( \delta_G = 0.1 \). The gamma is \( 0.9 \). There are carpets randomly placed in this environ-

When the robot traverses a carpet, it makes the carpet dirty. For each carpet, whether the robot is allowed to make it dirty is unknown to the robot. So each carpet corresponds to an unknown feature. All carpets are initially clean and the number of carpets varies in different experiments.

We compare several strategies and heuristics for query sele-

One, **optimal**, finds the exact optimal query policy: Consider a Query-MDP, \( M^Q \), where states are partitions of features. Actions are queries that probabilistically transit be-

The states where the switch is off; \( \gamma = 0.9 \). There are carpets randomly placed in this environ-

When the robot traverses a carpet, it makes the carpet dirty. For each carpet, whether the robot is allowed to make it dirty is unknown to the robot. So each carpet corresponds to an unknown feature. All carpets are initially clean and the number of carpets varies in different experiments.

We compare several strategies and heuristics for query sele-

One, **optimal**, finds the exact optimal query policy: Consider a Query-MDP, \( M^Q \), where states are partitions of features. Actions are queries that probabilistically transit be-

The states where the switch is on; \( \delta_G = 0.1 \). The gamma is \( 0.9 \). There are carpets randomly placed in this environ-

When the robot traverses a carpet, it makes the carpet dirty. For each carpet, whether the robot is allowed to make it dirty is unknown to the robot. So each carpet corresponds to an unknown feature. All carpets are initially clean and the number of carpets varies in different experiments.

We compare several strategies and heuristics for query sele-

One, **optimal**, finds the exact optimal query policy: Consider a Query-MDP, \( M^Q \), where states are partitions of features. Actions are queries that probabilistically transit be-

The states where the switch is off; \( \gamma = 0.9 \). There are carpets randomly placed in this environ-

When the robot traverses a carpet, it makes the carpet dirty. For each carpet, whether the robot is allowed to make it dirty is unknown to the robot. So each carpet corresponds to an unknown feature. All carpets are initially clean and the number of carpets varies in different experiments.

We compare several strategies and heuristics for query sele-

One, **optimal**, finds the exact optimal query policy: Consider a Query-MDP, \( M^Q \), where states are partitions of features. Actions are queries that probabilistically transit be-

The states where the switch is on; \( \delta_G = 0.1 \). The gamma is \( 0.9 \). There are carpets randomly placed in this environ-

When the robot traverses a carpet, it makes the carpet dirty. For each carpet, whether the robot is allowed to make it dirty is unknown to the robot. So each carpet corresponds to an unknown feature. All carpets are initially clean and the number of carpets varies in different experiments.

We compare several strategies and heuristics for query sele-

One, **optimal**, finds the exact optimal query policy: Consider a Query-MDP, \( M^Q \), where states are partitions of features. Actions are queries that probabilistically transit be-

The states where the switch is off; \( \gamma = 0.9 \). There are carpets randomly placed in this environ-

When the robot traverses a carpet, it makes the carpet dirty. For each carpet, whether the robot is allowed to make it dirty is unknown to the robot. So each carpet corresponds to an unknown feature. All carpets are initially clean and the number of carpets varies in different experiments.
heuristics that are based on the probability of existence of safe policies. probability-maximization finds the feature that, if known be to free, increases the probability of existence of safe policies the most: \( \arg \max_{\phi \in \Phi} p_F(\phi) \mathbb{P}[\tau; (\psi_\phi \in \Phi^F_L)] \). Similarly, probability-maximization-1 finds the feature that, if known to be locked, decreases the probability of existence of safe policies the most: \( \arg \max_{\phi \in \Phi} p_F(\phi) \mathbb{P}[\tau; (\psi_\phi \in \Phi^F_R)] \). Lastly, probability-maximization optimizes the joint objective of the previous two. \( \arg \max_{\phi \in \Phi} p_L(\phi) \mathbb{P}[\tau; (\psi_\phi \in \Phi^F_L)] + p_R(\phi) \mathbb{P}[\tau; (\psi_\phi \in \Phi^F_R)] \). We cannot adapt the algorithms in ZDS to this problem because they are designed for a minimax-regret setting and, crucially, depend on knowing a safe policy before querying.

We evaluate these selection strategies in the following experiments. We vary the numbers of unknown features (numbers of carpets in this example) in the first experiment and vary the distributions of \( p_F \) but fix the number of carpets in the second experiment.

**Experiment 1: Varying the number of carpets.** We first consider a uniformly random \( p_F \) setting with a varying number of carpets. The values of \( p_F(\phi), \phi \in \Phi^F_L \) are uniformly randomly generated between \([0,1]\). The numbers of carpets are 10, 12, 14. We run experiments for 200 trials for each data point. In each trial, the layout of walls and carpets are randomly generated. The performance of the heuristics and their computation times in domains with different numbers of carpets is shown in Figure 4.

As expected, with more carpets (unknown features), the robot needs more queries to find a safe policy or prove that none exists. Both \( h_{SC} \) and \( h_{ICR} \) have performance close to optimal while their computation is much cheaper than optimal (answering Q1). When the carpet number is 14, the optimal’s computation time per query is 41.41 ± 6.59 sec., which is out of the scope of the figure. Both \( h_{SC} \) and \( h_{ICR} \) find better queries than other candidate heuristics (answering Q2). We find \( h_{ICR} \) has performance closer to optimal compared with \( h_{SC} \) while using only slightly more computation time (answering Q3).

**Experiment 2: Varying \( p_F \).** In reality, it may not be the case that all unknown features have uniformly random probabilities of being free. In this experiment, we fix the number of carpets to be 14 and generate \( p_F \) in small intervals (Figure 5). The size of intervals are 0.5, that is, the intervals are \([0, 0.5], [0.1, 0.6], \ldots\). For clarity, we report a subset of algorithms in each figure.

We also compare to variations of \( h_{SC} \) that aim to cover \( S_{1IS} \) and \( S_{rel} \), respectively: \( h_{SC-IIS} \) queries about \( \arg \max_{\phi \in \Phi} \Delta_{1IS}(\phi(\psi))/|S_{1IS}| \); \( h_{SC-rel} \) queries about \( \arg \max_{\phi \in \Phi} \Delta_{rel}(\phi(\psi))/|S_{rel}| \). We have also described the most-likely-policy in the simple heuristics section. Finally, we also compare against the following
and IISs. Since the robot only needs to decide what is the next feature to query about, it may not be necessary to compute the exact $S_{IIS}$ and $S_{rel}$. It computes $S_{IIS}, S_{rel}$ for a limited time (5 seconds in our experiments) and stops, then selects queries based on the possibly incomplete sets (a similar idea is used in Regan and Boutilier (2010)). To determine $\top$ or $\bot$, the robot cannot simply check if $S_{IIS}$ or $S_{rel}$ is empty as in Algorithm SetCoverQuery (since both sets may be incomplete), but solves the LP problem in Eq. 1. Since we cannot afford to compute the optimal query policy, we use oracle as a baseline. It knows the true partition of the features and asks the minimum number of features possible to reach the correct outcome. We evaluate our heuristics in a $10 \times 10$ domain with 40 randomly placed carpets and 20 walls (Figure 6). We can see a similar pattern that $h_{ICR}$ is closest to optimal queries.

To summarize, we find both $h_{SC}$ and $h_{ICR}$ are robustly close to the optimal query when we vary the number of unknown features and distribution of $p_F$, while $h_{ICR}$ is closer to the optimal at the cost of slightly more computation. When $p_F$ is close to 0 or 1, some other candidate heuristics also have good performance (for example, using $h_{SC}$-IIS when $p_F$ values are large). These are also simpler cases since we can focus on covering either $S_{IIS}$ or $S_{rel}$.

**Conclusion and Future Work**

We have formally defined the problem of querying to find a safe policy, or proving none exists, in settings where unsafe side-effects are possible. Through a novel casting of that problem into a pair of set-cover problems, we have devised heuristics for a myopically-greedy approach that empirically perform nearly optimally without excessive computational costs, and outperform other candidate heuristics.

We have here assumed (as we did in ZDS) that features’ changeabilities are independent and feature values are static unless changed by the robot. Future work can relax these assumptions (some ideas were sketched in ZDS), consider other heuristics (possibly from ZDS), and explore multi-agent settings (e.g., one user overseeing multiple robots, one robot answering to multiple users). We also want to derive a performance bound for our heuristic similar to GK, and incorporate values of policies, so that the robot can bias its efforts towards finding safe policies with higher values.

**Acknowledgements.** Thanks to the anonymous reviewers. This work was supported in part by the Air Force Office of Scientific Research under grant FA9550-15-1-0039, and the Open Philanthropy Project to the Center for Human-Compatible AI. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

**References**

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

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *J. of Artificial Intelligence Research (JAIR)* 11(1):94.

Cakmak, M., and Thomaz, A. L. 2012. Designing robot learners that ask good questions. In *Proc. 7th Annual ACM/IEEE Int. Conf. on Human-Robot Interaction*, 17–24.

Chinneck, J. W. 2007. Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods, volume 118. Springer Science & Business Media.

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

Golovin, D., and Krause, A. 2011. Adaptive Submodularity: Theory and Applications in Active Learning and Stochastic Optimization. *J. of Artificial Intelligence Research* 42:427–486.

Le, T.; Tabakhi, A. M.; Tran-Thanh, L.; Yeoh, W.; and Son, T. C. 2018. Preference Elicitation with Interdependency and User Bother Cost. In *Proc. 17th Int. Conf. on Autonomous Agents and MultiAgent Systems*, 1459–1467.

Leike, J.; Martic, M.; Krakovna, V.; Ortega, P. A.; Everitt, T.; Lefrancq, A.; Orseau, L.; and Legg, S. 2017. AI safety gridworlds. *arXiv preprint arXiv:1711.09883*.

Mindermann, S.; Shah, R.; Gleave, A.; and Hadfield-Menell, D. 2018. Active Inverse Reward Design. *1st Workshop on Goal Specifications for Reinforcement Learning co-located with IJCAI 2018*.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1st edition.

Ramachandran, D., and Amir, E. 2007. Bayesian inverse reinforcement learning. In *Int. J. Conf. on Artificial Intelligence*, 2586–2591.

Regan, K., and Boutilier, C. 2009. Regret-based reward elicitation for Markov decision processes. In *Proc. Conf. on Uncertainty in Artificial Intelligence (UAI)*, 444–451.

Regan, K., and Boutilier, C. 2010. Robust policy computation in reward-uncertain MDPs using nondominated policies. In *Assoc. for Adv. of Artificial Intelligence (AAAI)*, 1127–1133.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

Turner, A. M.; Hadfield-Menell, D.; and Tadepalli, P. 2019. Conservative Agency. In *Proceedings of the Workshop on Artificial Intelligence Safety*, 29–36.

Williamson, D. P., and Shmoys, D. B. 2011. *The Design of Approximation Algorithms*. Cambridge University Press.

Zhang, S.; Durfee, E.; and Singh, S. 2018. Minimax-regret querying on side effects for safe optimality in factored Markov decision processes. In *Proc. 27th Int. J. Conf. on Artificial Intelligence (IJCAI)*, 4867–4873.