Minimax-Regret Querying on Side Effects for Safe Optimality in Factored Markov Decision Processes

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

Abstract

As it achieves a goal on behalf of its human user, an autonomous agent's actions may have side effects that change features of its environment in ways that negatively surprise its user. An agent that can be trusted to operate safely should thus only change features the user has explicitly permitted. We formalize this problem, and develop a planning algorithm that avoids potentially negative side effects given what the agent knows about (un)changeable features. Further, we formulate a provably minimax-regret querying strategy for the agent to selectively ask the user about features that it hasn't explicitly been told about. We empirically show how much faster it is than a more exhaustive approach and how much better its queries are than those found by the best known heuristic.

1 Introduction

We consider a setting where a human user tasks a computational agent with achieving a goal that may change one or more state features of the world (e.g., a housekeeping agent should change the state of the house floors and kitchen sink from dirty to clean). In the process of accomplishing the goal, the agent generally changes other features (e.g., its own position and power level, opening doors, moving furniture, scaring the cat). Some of these side-effects might be fine (even expected) by the user (e.g., moving), but others may be undesirable/unsafe (e.g., leaving doors open lets the cat roam/escape) even though they speed goal achievement (e.g., the agent's movement between rooms). Although the user tells the agent about some features that can be changed, as well as some to not change (e.g., don't knock over the priceless vase), the user often lacks the time, patience, or foresight to specify the changeability of every pertinent feature, and may incorrectly assume that the agent has commonsense (e.g., about cat behavior and the value of vases).

How can the agent execute a **safely**-optimal policy in such a setting? We conservatively assume that, to ensure safety, the agent should never side-effect a feature unless changing it is explicitly known to be fine. Hence, the agent could simply execute the best policy that leaves such features unchanged. However, no such policy might exist, and even if it does it might surprise the user as unnecessarily costly/inefficient.

Our focus is thus on how the agent can selectively query the user about the acceptability of changing features it hasn't yet been told about. We reject simply querying about every such feature, as this would be unbearably tedious to the user, and instead put the burden on the agent to limit the number and complexity of queries. In fact, in this paper we mostly focus on finding a single query about a few features that maximally improves upon the policy while maintaining safety.

Our three main contributions in this paper are: 1) We formulate (in Section 2) an AI safety problem of avoiding negative side-effects in factored MDPs. 2) We show (in Section 3) how to efficiently identify the set of **relevant** features, i.e., the set of features that could potentially be worth querying the user about. 3) We formulate (in Section 4) a minimax-regret criterion when there is a limit on the number of features the agent can ask about, and provide an algorithm that allows the agent to find the minimax-regret query by searching the query space with efficient pruning. We empirically evaluate our algorithms in a simulated agent navigation task, outline ongoing extensions/improvements, and contrast our work to prior work, in the paper's final sections.

2 Problem Definition

We illustrate our problem in a simulated agent gridworld-navigation domain, inspired by Amodei *et al.* [2016] and depicted in Figure 1, with doors, carpets, boxes, and a switch. The agent can open/close a door, move a box, traverse a carpet, and toggle the switch. Initially, the agent is in the bottom left corner; door d1 is open, d2 closed, the carpet clean, and the switch "on". The agent can move to an adjacent location vertically, horizontally, or diagonally. For simplicity, the transition function is assumed to be deterministic.

The user tasks the agent with turning off the switch as quickly as is *safely* possible. The quickest path (π_1) traverses the carpet, but this gets the carpet dirty and the agent doesn't know if that is allowed. The agent could instead enter the room through door d1 and spend time moving box b1 or b2 out of the way (π_2 or π_3 respectively), open door d2, and then go to the switch. However, boxes might contain fragile objects and should not be moved; the user knows each box's contents, but the agent doesn't. Or the agent could enter through door d1 and walk upwards (π_4) around all the boxes

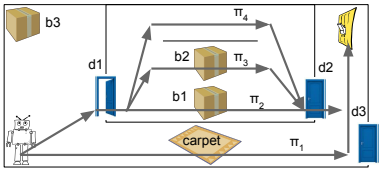

Figure 1: The robot navigation domain. The dominating policies (see Section 3) are shown as arrows.

and open door d2 to get to the switch. The user may or may not be okay with door d2 being opened. There are of course many other more circuitous paths not shown.

We model the domain as a factored Markov Decision Process (MDP) [Boutilier *et al.*, 1999]. An MDP is a tuple $\langle S, A, T, r, s_0, \gamma \rangle$, with state space S, action space A, and transition function T where $T(s'|s,a)$ 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 γ is the discount factor. Let $\pi : S \times A \rightarrow [0,1]$ be a policy. V^π is the expected cumulative reward by following policy π starting from s_0. In a factored MDP, a state is described in terms of values of various features (e.g., the agent's location, the current time, the status of each door, cleanliness of each carpet, position of each box), so the state space S is the cross-product of the values the features can take. The reward and transition functions are often also factored (e.g., the "toggle" action only changes the switch feature, leaving boxes, doors, and carpets unchanged). We will consistently use ϕ to denote one feature and Φ to denote a set of features. The agent knows the complete MDP model, but has incomplete knowledge about which features the user doesn't mind being changed. In our example, the user's goal implies the agent's location is changeable, as is the switch, but the agent is uncertain about side-effecting boxes, doors, and carpets.

In general, the user could dictate that a feature can only be changed among restricted values (e.g., boxes can only be moved a short distance) and/or dependent on other features' values (e.g., interior doors can be left open as long as exterior doors (like d3) stay closed). We briefly return to this issue later (Section 6), but for simplicity assume here that the agent can partition the features into the following sets:

- $\Phi_F^{\mathbf{A}}$: The *free-features*. The agent knows that these features are freely changeable (e.g., its location).
- $\Phi_L^{\mathbf{A}}$: The *locked-features*. The agent knows it should never change any of these features.
- $\Phi_?^{\mathbf{A}}$: The *unknown-features*. These are features that the agent doesn't (initially) know whether the user considers freely changeable or locked.

The user similarly partitions the features, but only into the sets $\Phi_L^{\mathbf{U}}$ and $\Phi_F^{\mathbf{U}}$. We assume that the agent's knowledge, while generally incomplete ($\Phi_?^{\mathbf{A}} \neq \emptyset$), is consistent with that of the user. That is, $\Phi_F^{\mathbf{A}} \subseteq \Phi_F^{\mathbf{U}}$ and $\Phi_L^{\mathbf{A}} \subseteq \Phi_L^{\mathbf{U}}$.

Defining & Finding Safely-Optimal Policies. Our conservative safety assumption means the agent should treat unknown features as if they are locked, until it explicitly knows otherwise. It should thus find an optimal policy that never visits a state where a feature in $\Phi_L^{\mathbf{A}} \cup \Phi_?^{\mathbf{A}}$ is changed, which we call a **safely-optimal policy**. We can use linear programming with constraints that prevent the policy from visiting states with changed values of locked or unknown features:

$$\max_x \sum_{s,a} x(s,a) r(s,a) \text{ s.t.} \tag{1}$$

$$\forall s' \in S, \sum_{a'} x(s',a') = \gamma \sum_{s,a} x(s,a) T(s'|s,a) + \delta(s',s_0)$$

$$\forall s \in S_{\Phi_L^{\mathbf{A}} \cup \Phi_?^{\mathbf{A}}}, \forall a \in A, x(s,a) = 0$$

The control variables $x : S \times A \rightarrow \mathbb{R}$ in the linear program are occupancy measures, i.e., $x(s,a)$ is the expected discounted number of times that state action pair s, a is visited by the agent's policy, S_Φ is the set of states where one or more features in Φ have different values from the initial state, and $\delta(s,s') = 1$ if $s = s'$ and is zero otherwise.

The above linear program does not allow any locked or unknown feature to be changed and is guaranteed to produce a safely-optimal policy (when one exists). This linear programming approach, while straightforward, can be intractable for large MDPs. Alternative approaches could directly encode the safety constraints into the transition function (e.g., by removing unsafe action choices in specific states), or into the reward function (heavily penalizing reaching unsafe states). Approximate methods, like feature-based approximate linear programming [Dolgov and Durfee, 2006] or constrained policy optimization [Achiam *et al.*, 2017], can apply to larger problems, but may not guarantee safety or optimality (or both). We return to these concerns in Section 6.

3 Querying Relevant Unknown-Features

In our setting the only way for the agent to determine whether an unknown feature is freely changeable is to query the user. Thus, hereafter our focus is on how the agent can ask a good query about a small number, k, of features. Our solution is to first prune from $\Phi_?^{\mathbf{A}}$ features that are guaranteed to not be relevant to ask (this section), and then efficiently finding the best (minimax-regret) k-sized subset of the relevant features to query (Section 4). Until Section 6, we assume the changeabilities of features are independent, i.e., when the agent asks about some features in $\Phi_?^{\mathbf{A}}$, the user's response does not allow it to infer the changeability of other features in $\Phi_?^{\mathbf{A}}$.

Intuitively, when is a feature in $\Phi_?^{\mathbf{A}}$ relevant to the agent's planning of a safely-optimal policy? In the navigation domain, if the agent plans to take the quickest path to the switch (π_1 in Figure 1), it will change the state of the carpet (from clean to dirty). The carpet feature is thus *relevant* since the agent would change it if permitted. If the carpet can't be changed but door d2 can, the agent would follow (in order of preference) policy π_2, π_3, or π_4, so d2, b1, and b2 are relevant. Box b3 and door d3 are *irrelevant*, however, since no matter which (if any) other features are changeable, an optimal policy would never change them. Thus, an unknown feature is relevant when under some circumstance (some answer to some query) the agent's optimal policy would side-effect that feature. Such policies are *dominating policies*.

A **dominating policy** is a safely-optimal policy for the circumstance where the unknown features $\Phi_?^{\mathbf{A}}$ are partitioned

Algorithm *DomPolicies*

1: $\Gamma' \leftarrow \emptyset$ ▷ the initial set of dominating policies
2: $\Phi'_{rel} \leftarrow \emptyset$ ▷ the initial set of relevant features
3: $checked \leftarrow \emptyset$ ▷ It contains $\Phi \subseteq \Phi'_{rel}$ we have examined so far.
4: $\beta \leftarrow \emptyset$ ▷ a pruning rule
5: $agenda \leftarrow powerset(\Phi'_{rel}) \setminus checked$
6: **while** $agenda \neq \emptyset$ **do**
7: $\Phi \leftarrow$ least-cardinality element of $agenda$
8: **if** $satisfy(\Phi, \beta)$ **then**
9: (get safely-optimal policy with Φ locked)
10: $\pi \leftarrow \arg\max_{\pi' \in \Pi_\Phi} V^{\pi'}$ ▷ by solving Eq. 1
11: **if** π exists **then**
12: $\Gamma' \leftarrow \Gamma' \cup \{\pi\}$; add $(\Phi, \Phi_{rel}(\pi))$ to β
13: **else** add $(\Phi, \emptyset)$ to β
14: $\Phi'_{rel} \leftarrow \Phi'_{rel} \cup \Phi_{rel}(\pi)$
15: $checked \leftarrow checked \cup \{\Phi\}$
16: $agenda \leftarrow powerset(\Phi'_{rel}) \setminus checked$
17: return Γ', Φ'_{rel}

into locked and changeable subsets. We denote the set of dominating policies by

$$\Gamma = \left\{ \arg\max_{\pi \in \Pi_\Phi} V^\pi : \forall \Phi \subseteq \Phi^{\mathbf{A}}_? \right\}, \quad (2)$$

where Π_Φ is the set of policies that do not change unknown features $\Phi \subseteq \Phi_?$ as well as any locked features (meaning that $\Phi_F \cup (\Phi_? \setminus \Phi)$ *are* changeable). We denote the unknown-features side-effected by policy π by $\Phi_{rel}(\pi)$. For a set of policies Π, $\Phi_{rel}(\Pi) = \cup_{\pi \in \Pi} \Phi_{rel}(\pi)$. The set $\Phi_{rel}(\Gamma)$, abbreviated Φ_{rel}, is thus the set of **relevant (unknown-) features** to consider querying about.

Instead of finding the safely-optimal policies for all exponentially (in $|\Phi^{\mathbf{A}}_?|$) many subsets $\Phi \subseteq \Phi^{\mathbf{A}}_?$ with Equation 2, we contribute Algorithm *DomPolicies* (see pseudocode) that finds dominating policies incrementally (and in practice more efficiently) by constructing the sets of relevant features and dominating policies simultaneously. In each iteration, it examines a new subset of relevant features, Φ, and, if Φ isn't pruned (as described later), finds the safely-optimal policy with Φ being locked (Line 7). It then adds $\Phi_{rel}(\pi)$, the features changed by π, to Φ'_{rel}. It repeats this process until Φ'_{rel} stops growing and all subsets of Φ'_{rel} are examined. For example, in the navigation problem (Figure 1), the first added policy is the safely-optimal policy assuming no unknown features are locked, which is π_1. Now $\Gamma' = \{\pi_1\}$ and thus $\Phi'_{rel} = \{carpet\}$. It iterates, treating the carpet as locked, and updates Γ' to $\{\pi_1, \pi_2\}$ and thus $\Phi'_{rel} = \{carpet, b1, d2\}$. Iterating again, it locks subsets of Φ'_{rel}, finding π_3 for subset $\{carpet, b1\}$. After finding π_4, it terminates.

In Line 10, the algorithm finds the constrained optimal policy (in our implementation using Eq. 1), and in the worst case, would do this $2^{|\Phi_{rel}|}$ times. Fortunately, the complexity is exponential in the number of *relevant* features, which as we have seen in some empirical settings can be considerably smaller than the number of unknown features ($|\Phi^{\mathbf{A}}_?|$). Furthermore, the efficiency can be improved with our pruning rule (line 8):

$$satisfy(\Phi, \beta) := \neg \exists_{(L,R) \in \beta} (L \subseteq \Phi \ \wedge \ \Phi \cap R = \emptyset)$$

β is a history of ordered pairs, (L, R), of disjoint sets of unknown features. Before *DomPolicies* computes a policy for its agenda element Φ, if a pair (L, R) is in β, such that $L \subseteq \Phi$ (a dominating policy has been found when locking subset of features in Φ), and that dominating policy's relevant features R don't intersect with Φ, then Φ's dominating policy has already been found. In our running example, for instance, initially $\Phi'_{rel} = \emptyset$. π_1 is the optimal policy and the algorithm adds pair $(\emptyset, \{carpet\})$ to β. When larger subsets are later considered (note the agenda is sorted by cardinality), feature sets that do not contain $carpet$ are pruned by β. In this example, β prunes 11 of the 16 subsets of the 4 relevant features.

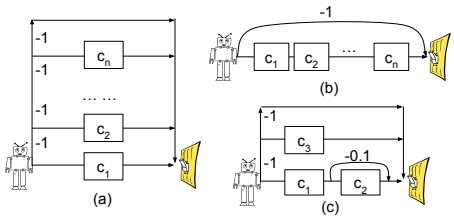

Figure 2: Example domains used in text. ($n > 2$)

Consider the examples in Figure 2. To reach the switch, the agent could traverse zero or more carpets (all with unknown changeability), and rewards are marked on the edges. (a) and (b) only need to compute policies linear in the number of relevant features: Only $n+1$ dominating safely-optimal policies are computed for Figure 2(a) (for $\Phi = \emptyset, \{c_1\}, \{c_1, c_2\}, \dots$), and Figure 2(b) (for $\Phi = \emptyset, \{c_1\}, \{c_2\}, \dots, \{c_n\}$). Figure 2(c) computes policies for only half of the subsets.

Theorem 1. *The set of policies returned by Algorithm DomPolicies is the set of all dominating policies.*

Proof. Let $\pi \in \Gamma$ be the optimal policy with unknown features L locked. Γ' is returned by *DomPolicies*. We denote $L \cap \Phi_{rel}(\Gamma')$ as A and $L \setminus \Phi_{rel}(\Gamma')$ as B. For a proof by contradiction, assume $\pi \notin \Gamma'$. Then $B \neq \emptyset$. Otherwise, if $B = \emptyset$ (or equivalently, $A = L$), then L is a subset of $\Phi_{rel}(\Gamma')$ and π would have been added to Γ'. Let π' be the optimal policy with A locked. Since $A \subseteq \Phi_{rel}(\Gamma')$, we know $\pi' \in \Gamma'$. We observe that π' does not change any features in B (otherwise features in B would show up in $\Phi_{rel}(\Gamma')$). So π' is also the optimal policy with features $A \cup B = L$ locked. So $\pi = \pi'$, which is an element of Γ'. □

4 Finding Minimax-Regret Queries

With *DomPolicies*, the agent need only query the user about relevant features. But it could further reduce the user's burden by being selective about *which* relevant features it asks about. In our running example (Figure 1), for instance, *DomPolicies* removes b3 and d3 from consideration, but also intuitively the agent should only ask about b1 or b2 if d2 is changeable. By iteratively querying, accounting for such dependencies (which can be gleaned from β in *DomPolicies*) and updating the relevant feature set, the agent can stop querying as

soon as it finds (if one exists) a safe policy. For example, say it asks about the carpet and d2, and is told d2 (but not the carpet) is changeable. Now it has a safely-optimal policy, π_4, given its knowledge, and could stop querying. But π_4 is the worst safe policy. Should it ask about boxes?

That is the question we focus on here: How should an agent query to try to find a *better* safely-optimal policy than the one it already knows about. Specifically, we consider the setting where the agent is permitted to interrupt the user just *once* to improve its safely-optimal policy, by asking a single query about at most k unknown features. For each feature the user will reply whether or not it is in $\Phi_F^{\mathbf{U}}$. Formally, Φ_q is a k-*feature query* where $\Phi_q \subseteq \Phi_{rel}$ and $|\Phi_q| = k$. The post-response utility when the agent asks query Φ_q, and $\Phi_c \subseteq \Phi_?^{\mathbf{A}}$ are actually changeable, is the value of the safely-optimal policy after the user's response:

$$u(\Phi_q, \Phi_c) = \max_{\pi \in \Pi_{\Phi_?^{\mathbf{A}} \setminus (\Phi_q \cap \Phi_c)}} V^{\pi}. \quad (3)$$

Recall the agent can only safely change features it queries about that the user's response indicates are changeable ($\Phi_q \cap \Phi_c$). What would be the agent's regret if it asks a k-feature query Φ_q rather than a k-feature query $\Phi_{q'}$? We consider the circumstance where a set of features Φ_c are changeable and under which the difference between the utilities of asking Φ_q and $\Phi_{q'}$ is maximized. We call this difference of utilities the *pairwise maximum regret* of queries Φ_q and $\Phi_{q'}$, defined below in a similar way to Regan and Boutilier [2010]:

$$PMR(\Phi_q, \Phi_{q'}) = \max_{\Phi_c \subseteq \Phi_?^{\mathbf{A}}} (u(\Phi_{q'}, \Phi_c) - u(\Phi_q, \Phi_c)). \quad (4)$$

The **maximum regret**, denoted by MR, of query Φ_q is determined by the $\Phi_{q'}$ that maximizes $PMR(\Phi_q, \Phi_{q'})$:

$$MR(\Phi_q) = \max_{\Phi_{q'} \subseteq \Phi_{rel}, |\Phi_{q'}| = k} PMR(\Phi_q, \Phi_{q'}). \quad (5)$$

The agent should ask the minimax-regret (k-feature) query:

$$\Phi_q^{MMR} = \arg \min_{\Phi_q \subseteq \Phi_{rel}, |\Phi_q| = k} MR(\Phi_q). \quad (6)$$

The rationale of the minimax-regret criterion is as follows. Whenever the agent considers a query Φ_q, there could exist a query $\Phi_{q'}$ that is better than Φ_q under some true changeable features Φ_c. The agent focuses on the worst case Φ_c, where the difference between the utility of Φ_q and the best query $\Phi_{q'}$ that could be asked is maximized. The agent uses adversarial reasoning to efficiently find the worst-case Φ_c: Given that it is considering query Φ_q, it asks what query $\Phi_{q'}$ and set of features Φ_c an *imaginary adversary* would pick to maximize the gap between the utilities of $\Phi_{q'}$ and Φ_q under Φ_c (that is, $u(\Phi_{q'}, \Phi_c) - u(\Phi_q, \Phi_c)$). The agent wants to find a query Φ_q that minimizes the worst case (maximum gap).

Under the definition of MR, the agent, reasoning as if it were its imaginary adversary, must find the maximizing $\Phi_{q'}$ and Φ_c. However, we can simplify the definition so that it only needs to find maximizing Φ_c. Note that since an imaginary adversary chooses both $\Phi_{q'}$ and Φ_c, it wants to make sure that $\Phi_c \subseteq \Phi_{q'}$, which means that it does not want the features not in $\Phi_{q'}$ to be changeable. We then observe that

$$MR(\Phi_q) = \max_{\substack{\pi' \in \Gamma: \\ \Phi_{rel}(\pi') \leq k}} (V^{\pi'} - \max_{\pi \in \Pi_{\Phi_?^{\mathbf{A}} \setminus \{\Phi_q \cap \Phi_{rel}(\pi')\}}} V^{\pi}) \quad (7)$$

We call the π' maximizing Eq. 7 the **adversarial policy** when the agent asks query Φ_q, denoted by $\pi_{\Phi_q}^{MR}$. With Eq. 7, the agent can compute MR based on the set of dominating policies (which *DomPolicies* already found), rather than the (generally much larger) powerset of the relevant features in Eq. 5.

While using Eq. 7 is faster than Eq. 5, the agent still needs to do this computation for every possible query (Eq. 6) of size k. We contribute two further ways to improve the efficiency. First, we may not need to consider all relevant features if we can only ask about k of them. If a subset of relevant features satisfies the condition in Theorem 2, then we call it a set of *sufficient features*, because a minimax-regret k-feature query from that set is a globally minimax-regret k-feature query. Second, we introduce a pruning rule that we call *query dominance* in Theorem 3 to safely eliminate considering queries that cannot be better than ones already evaluated.

The following theorem shows that if we can find any subset Φ of Φ_{rel} such that for all k-feature subsets of Φ as queries, the associated adversarial policy's relevant features are contained in Φ, then the minimax regret query found by restricting queries to be subsets of Φ will also be a minimax regret query found by considering all queries in Φ_{rel}. Such a (non-unique) set Φ will be referred to as a *sufficient feature set* (for the purpose of finding minimax regret queries).

Theorem 2. (Sufficient Feature Set) *For any set of* $\geq k$ *features* Φ*, if for all* $\Phi_q \subseteq \Phi$*,* $|\Phi_q| = k$*, we have* $\Phi_{rel}(\pi_{\Phi_q}^{MR}) \subseteq \Phi$*, then* $\min_{\Phi_q \subseteq \Phi, |\Phi_q| = k} MR(\Phi_q) = \min_{\Phi_q \subseteq \Phi_{rel}, |\Phi_q| = k} MR(\Phi_q)$.

Proof Sketch: If a set of features Φ, $|\Phi| \geq k$, fails to include some features in Φ_q^{MMR}, when we query some k-subset of Φ, the adversarial policy should change some of the features in $\Phi_q^{MMR} \setminus \Phi$. Otherwise, querying about $\Phi_q^{MMR} \setminus \Phi$ does not reduce the maximum regret. Then $\Phi_q^{MMR} \setminus \Phi$ are not necessary to be included in Φ_q^{MMR}.

Given a set of sufficient features, the following theorem shows that it may not be necessary to compute the maximum regrets for all k-subsets to find the minimax-regret query.

Theorem 3. (Query Dominance) *For any pair of queries* Φ_q *and* $\Phi_{q'}$*, if* $\Phi_{q'} \cap \Phi_{rel}(\pi_{\Phi_q}^{MR}) \subseteq \Phi_q \cap \Phi_{rel}(\pi_{\Phi_q}^{MR})$*, then* $MR(\Phi_{q'}) \geq MR(\Phi_q)$.

Proof. Observe that $MR(\Phi_{q'})$
$\geq V^{\pi_{\Phi_q}^{MR}} - \max_{\pi' \in \Pi_{\Phi_?^{\mathbf{A}} \setminus (\Phi_{q'} \cap \Phi_{rel}(\pi_{\Phi_q}^{MR}))}} V^{\pi'}$
$\geq V^{\pi_{\Phi_q}^{MR}} - \max_{\pi' \in \Pi_{\Phi_?^{\mathbf{A}} \setminus (\Phi_q \cap \Phi_{rel}(\pi_{\Phi_q}^{MR}))}} V^{\pi'} = MR(\Phi_q)$ □

We denote the condition $\Phi_{q'} \cap \Phi_{rel}(\pi_{\Phi_q}^{MR}) \subseteq \Phi_q \cap \Phi_{rel}(\pi_{\Phi_q}^{MR})$ by $dominance(\Phi_q, \Phi_{q'})$. To compute dominance, we only need to store $\Phi_{rel}(\pi_{\Phi_q}^{MR})$ for all Φ_q we have considered.

Algorithm *MMRQ-k* below provides pseudocode for finding a minimax-regret k-feature query; it takes advantage of both the notion of a sufficient-feature-set as well as query dominance to reduce computation significantly relative to the brute-force approach of searching over all k-feature queries (subsets) of the relevant feature set.

Algorithm *MMRQ-k*

1: $\Phi_q \leftarrow$ an initial k-feature query
2: $checked \leftarrow \emptyset$; $evaluated \leftarrow \emptyset$
3: $\Phi_{suf} \leftarrow \Phi_q$
4: $agenda \leftarrow \{\Phi_q\}$
5: **while** $agenda \neq \emptyset$ **do**
6: $\quad \Phi_q \leftarrow$ an element from $agenda$
7: $\quad$ **if** $\neg\exists_{\Phi_{q'} \in evaluated} dominance(\Phi_{q'}, \Phi_q)$ **then**
8: $\quad\quad$ Compute $MR(\Phi_q)$ and $\pi^{MR}_{\Phi_q}$
9: $\quad\quad \Phi_{suf} \leftarrow \Phi_{suf} \cup \Phi_{rel}(\pi^{MR}_{\Phi_q})$
10: $\quad\quad evaluated \leftarrow evaluated \cup \{\Phi_q\}$
11: $\quad checked \leftarrow checked \cup \{\Phi_q\}$
12: $\quad agenda \leftarrow all\_k\_subsets\_of(\Phi_{suf}) \setminus checked$
13: return $\arg\min_{\Phi_q \in evaluated} MR(\Phi_q)$

Intuitively, the algorithm keeps augmenting the set of features Φ_{suf}, which contain the features in the queries we have considered and the features changed by their adversarial policies, until it becomes a sufficient feature set. $agenda$ keeps track of k-subsets in Φ_{suf} that we have not yet evaluated. According to Theorem 2, we can terminate the algorithm when $agenda$ is empty (Line 5). We also use Theorem 3 to filter out queries which we know are not better than the ones we have found already (Line 7). Note that an initial Φ_{suf} needs to be chosen, which can be arbitrary. Our implementation initializes Φ_q with the Chain of Adversaries heuristic (Section 5).

To illustrate when Algorithm *MMRQ-k* can and can't prune suboptimal queries and thus gain efficiency, consider finding the minimax-regret 2-feature query in Figure 2 (a), which should be $\{c_1, c_2\}$. If the agent considers a query that does not include c_1, the adversarial policy would change c_1, adding c_1 to Φ_{suf}. If the agent considers a query that includes c_1 but not c_2, the adversarial policy would change c_2, adding c_2 to Φ_{suf}. When the agent asks $\{c_1, c_2\}$, the adversarial policy changes c_3 (adversarially asserting that c_1, c_2 are locked), so c_3 is added to Φ_{suf}. With $\{c_1, c_2, c_3\} \subseteq \Phi_{suf}$, the condition in Theorem 2 holds and the $n-3$ other features can be safely ignored. The minimax-regret query constituted by features in Φ_{suf} is $\{c_1, c_2\}$. However, in Figure 2 (b), $\Phi_{suf} = \Phi_{rel}$, and all $\binom{|\Phi_{rel}|}{2}$ would be evaluated.

5 Empirical Evaluations

We now empirically confirm that Algorithm *MMRQ-k* finds a minimax-regret query, and its theoretically sound Sufficient-Feature-Set and Query-Dominance based improvements can indeed pay computational dividends. We also compare our *MMRQ-k* algorithm to baseline approaches and the **Chain of Adversaries (CoA)** heuristic [Viappiani and Boutilier, 2009] adapted to our setting. Algorithm *CoA* begins with $\Phi_{q_0} = \emptyset$ and improves this query by iteratively computing:

$$\tilde{\pi} \leftarrow \arg \max_{\pi' \in \Gamma : |\Phi_{rel}(\pi') \cup \Phi_{q_i}| \leq k} (V^{\pi'} - \max_{\pi \in \Pi_{\Phi^{\mathbf{A}}_?} \setminus \{\Phi_{q_i} \cap \Phi_{rel}(\pi')\}} V^{\pi})$$

$$\Phi_{q_{i+1}} \leftarrow \Phi_{q_i} \cup \Phi_{rel}(\tilde{\pi}).$$

The algorithm stops when $|\Phi_{q_{i+1}}| = k$ or $\Phi_{q_{i+1}} = \Phi_{q_i}$. Although Algorithm *CoA* greedily adds features to the query to reduce the maximum regret, unlike *MMRQ-k* it does not guarantee finding the minimax-regret query. For example, in Figure 2(c), when $k = 2$, CoA first finds the optimal policy, which changes $\{c_1, c_2\}$, and returns that as a query, while the minimax-regret query is $\{c_1, c_3\}$.

We compare the following algorithms in this section: 1. Brute force (rel. feat.) uses Algorithm *DomPolicies* to find all relevant features first and evaluates all k-subsets of the relevant features. 2. Algorithm *MMRQ-k*. 3. Algorithm *CoA*. 4. Random queries (rel. feat.), which contain k uniformly randomly chosen relevant features. 5. Random queries, which contain k uniformly randomly chosen unknown features, without computing relevant features first. 6. No queries (equivalently vacuous queries).

We evaluate the algorithms' computation times and the quality of the queries they find, reported as the *normalized* MR to capture their relative performance compared to the best and the worst possible queries. That is, the normalized MR of a query Φ_q is defined as $(MR(\Phi_q) - MR(\Phi_q^{MMR}))/(MR(\Phi_{q_\perp}) - MR(\Phi_q^{MMR}))$, where $\Phi_{q_\perp}$ is a vacuous query, containing k features that are irrelevant and/or already known. The normalized MR of a minimax-regret query is 0 and that of a vacuous query is 1.

Navigation. As illustrated in Figure 3, the robot starts from the left-bottom corner and is tasked to turn off a switch at the top-right corner. The size of the domain is 6×6. The robot can move one step north, east or northeast at each time step. It stays in place if it tries to move across a border. The discount factor is 1. Initially, 10 clean carpets are uniformly randomly placed in the domain (the blue cells). In any cell without a carpet, the reward is set to be uniformly random in $[-1, 0]$, and in a cell with a carpet the reward is 0. Hence, the robot will generally prefer to walk on a carpet rather than around it. The state of each carpet corresponds to one feature. The robot is uncertain about whether the user cares about whether any particular carpet gets dirty, so all carpet features are in $\Phi^{\mathbf{A}}_?$. The robot knows that its own location and the state of the switch are in $\Phi^{\mathbf{A}}_F$. Since *MMRQ-k* attempts to improve on an existing safe policy, the left column and the top row never have carpets to ensure at least one safe path to the switch (the dotted line). The robot can ask one k-feature query before it takes any physical actions. We report results on 1500 trials. The only difference between trials are the locations of carpets, which are uniformly randomly placed.

First, we compare the brute force method to our *MMRQ-k* algorithm. We empirically confirm that in all cases *MMRQ-k* finds a minimax regret query, matching Brute Force performance. We also see that brute force scales poorly as k grows while *MMRQ-k*, benefitting from Theorems 2 and 3, is more computationally efficient (Figure 4).

We then want to see if and when *MMRQ-k* outperforms other candidate algorithms. In Figure 4, when k is small, the greedy choice of *CoA* can often find the best features to add to the small query, but as k increases, *CoA* suffers from being too greedy. When k is large (approaches the number of unknown features), being selective is less important and all methods find good queries. We also consider how $|\Phi_{rel}|$ affects the performance (Figure 5). When $|\Phi_{rel}|$ is smaller

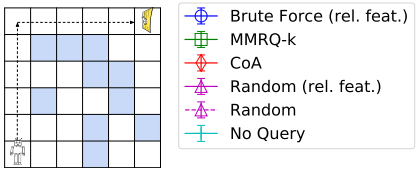

Figure 3: Office navigation and legend for following figures.

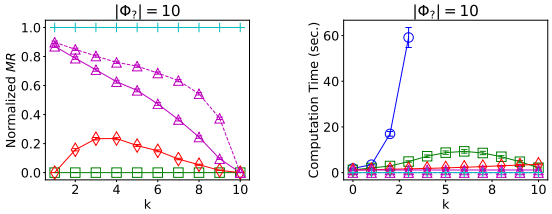

Figure 4: Normalized maximum MR vs. k. $|\Phi_?| = 10$. Brute force computation time is only shown for $k = 0, 1, 2, 3$.

than k, a k-feature query that contains all relevant features is optimal. All algorithms would find an optimal query except Random (which selects from all unknown features) and No Queries. (The error bars are larger for small $|\Phi_{rel}|$ since more rarely are only very few features relevant.) When $|\Phi_{rel}|$ is slightly larger than k, even a random query may be luckily a minimax-regret query, and *CoA* unsurprisingly finds queries close to the minimax-regret queries. However, when $|\Phi_{rel}|$ is much larger than k, the gap between *MMRQ-k* and other algorithms is larger. In summary, *MMRQ-k*'s benefits increase with the opportunities to be selective (larger $\binom{|\Phi_{rel}|}{k}$).

We have also experimented with *expected* regret given a probabilistic model of how the user will answer queries. For example, if the user has probability p of saying an unknown feature is free ($1 - p$ it's locked), then as expected, when p is very low, querying rarely helps, so using *MMRQ-k* or *CoA* matters little, and as p nears 1, *CoA* 's greedy optimism pays off to meet *MMRQ-k* 's minimax approach. But, empirically, MMRQ-k outperforms *CoA* for non-extreme values of p.

6 Extensions and Scalability

We now briefly consider applying our algorithms to larger, more complicated problems. As mentioned in Section 2, features' changeabilities might be more nuanced, with restrictions on what values they can take, individually or in combination. An example of the latter from Fig. 1 is where doors are *revertible* features, which means their changeability is dependent on the "time" feature: they are freely changeable except that by the time the episode ends they need to revert to their initial values. This expands the set of possible feature queries (e.g., asking if d2 is changeable differs from asking if it is revertible). In our experiments, this change accentuates the advantages of *MMRQ-k* over *CoA*: *CoA* asks (is-d2-locked, is-d2-revertible), hoping to hear "no" to both and follow a policy going through d2 without closing it. *MMRQ-k* asks (is-d2-locked, is-carpet-locked): since closing d2 only takes an extra time step, it is more valuable to know if the carpet is locked than if d2 can be left open.

More nuanced feature changeability means *DomPolicies* will have to find policies for the powerset of every relevant

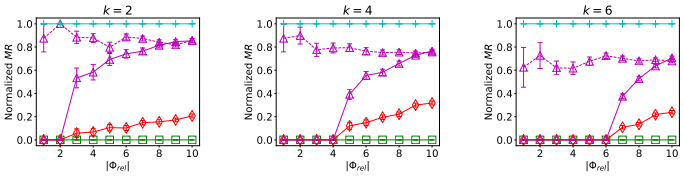

Figure 5: Normalized MR vs. the number of relevant features. $|\Phi_?| = 10$ and $k = 2, 4, 6$.

combination of features' values (in the worst case the size of the state space). One option is to ignore nuances in ways that maintain safety (e.g., treat a feature as revertible even if sometimes it can be left changed) and solve such a safe abstraction of the problem. Or one could abstract based on guaranteed correlations in feature changeabilities (e.g., if all boxes have the same changeability then ignore asking about all but one). Another option is to find only a subset of dominating policies, for example by using knowlege of k to avoid finding dominating policies that would change more unknown features than could be asked about anyway. And, or course, as mentioned before, finding approximately-safely-optimal policies in *DomPolicies* Line 10 would help speed the process (and might be the only option for larger problem domains).

Fortunately, such abstractions, heuristics, and approximations do not undermine safety guarantees. Recall that *DomPolicies* and *MMRQ-k* are finding a query, not the agent's final policy: the safety of the agent depends on how it finds the policy it executes, not on the safety of policies for hypothetical changeability conditions. However, as coarser abstractions, heuristics, and approximations are employed in our algorithms, the queries found can increasingly deviate from the minimax-regret optima. Fortunately, if the agent begins with a safely-optimal policy, "quick and dirty" versions of our methods can never harm it (they just become less likely to help). And if it begins without such a policy, such versions of our methods might not guide querying well, but by eventually asking about every unknown feature (in the worst case) a safe policy will still be found (if one exists).

7 Related Work & Summary

Amodei *et al.* [2016] address the problem of avoiding negative side-effects by penalizing all side-effects while optimizing the value. In our work, we allow the agent to communicate with the user. Safety is also formulated as resolving reward uncertainty [Amin *et al.*, 2017; Hadfield-Menell *et al.*, 2017], following imperfectly-specified instructions [Milli *et al.*, 2017], and learning safe states [Laskey *et al.*, 2016]. Safety issues also appear in exploration [Hans *et al.*, 2008; Moldovan and Abbeel, 2012; Achiam *et al.*, 2017]. Here we only provide a brief survey on safety in MDPs. Leike *et al.* [2017], Amodei *et al.* [2016] and Garcıa and Fernández [2015] are more thorough surveys.

There are problems similar to finding safely-optimal policies, which find policies that satisfy some constraints/commitments or maximize the probability of reaching a goal state [Witwicki and Durfee, 2010; Teichteil-Königsbuch, 2012; Kolobov *et al.*, 2012]. There are also other works using minimax-regret and policy dominance [Regan and Boutilier, 2010; Nilim and El Ghaoui, 2005]. and

querying to resolve uncertainty [Weng and Zanuttini, 2013; Regan and Boutilier, 2009; Cohn *et al.*, 2011; Zhang *et al.*, 2017]. We've combined and customized these ideas to find a provably minimax-regret k-element query.

In summary, we addressed the problem of an agent selectively querying a user about what features can be safely side-effected. We borrowed existing ideas from the literature about dominating policies and minimax regret, wove them together in a novel way, and streamlined the resulting algorithms to improve scalability while maintaining safe optimality.

Acknowledgements: Thanks to the anonymous reviewers. Supported in part by the US Air Force Office of Scientific Research, under grant FA9550-15-1-0039. Satinder Singh also acknowledges support from the Open Philanthropy Project to the Center for Human-Compatible AI.

References

[Achiam *et al.*, 2017] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 22–31, 2017.

[Amin *et al.*, 2017] Kareem Amin, Nan Jiang, and Satinder Singh. Repeated inverse reinforcement learning. In *Adv. in Neural Info. Proc. Sys. (NIPS)*, pages 1813–1822, 2017.

[Amodei *et al.*, 2016] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

[Boutilier *et al.*, 1999] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *J. of Artificial Intelligence Research (JAIR)*, 11(1):94, 1999.

[Cohn *et al.*, 2011] Robert Cohn, Edmund Durfee, and Satinder Singh. Comparing action-query strategies in semi-autonomous agents. In *Int. Conf. on Autonomous Agents and Multiagent Systems*, pages 1287–1288, 2011.

[Dolgov and Durfee, 2006] Dmitri A. Dolgov and Edmund H. Durfee. Symmetric approximate linear programming for factored MDPs with application to constrained problems. *Annals of Mathematics and Artificial Intelligence*, 47(3):273–293, Aug 2006.

[Garcıa and Fernández, 2015] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *J. of Machine Learning Research (JMLR)*, 16(1):1437–1480, 2015.

[Hadfield-Menell *et al.*, 2017] Dylan Hadfield-Menell, Smitha Milli, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Inverse reward design. In *Adv. in Neural Info. Processing Systems (NIPS)*, pages 6749–6758, 2017.

[Hans *et al.*, 2008] Alexander Hans, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft. Safe exploration for reinforcement learning. In *Euro. Symp. on Artificial Neural Networks (ESANN)*, pages 143–148, 2008.

[Kolobov *et al.*, 2012] Andrey Kolobov, Mausam, and Daniel S. Weld. A theory of goal-oriented MDPs with dead ends. In *Proc. Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 438–447, 2012.

[Laskey *et al.*, 2016] Michael Laskey, Sam Staszak, Wesley Yu-Shu Hsieh, Jeffrey Mahler, Florian T Pokorny, Anca D Dragan, and Ken Goldberg. SHIV: Reducing supervisor burden in DAgger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 462–469, 2016.

[Leike *et al.*, 2017] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. AI safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

[Milli *et al.*, 2017] Smitha Milli, Dylan Hadfield-Menell, Anca D. Dragan, and Stuart J. Russell. Should robots be obedient? In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 4754–4760, 2017.

[Moldovan and Abbeel, 2012] Teodor M. Moldovan and Pieter Abbeel. Safe exploration in Markov decision processes. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 1711–1718, 2012.

[Nilim and El Ghaoui, 2005] Arnab Nilim and Laurent El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

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

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

[Teichteil-Königsbuch, 2012] Florent Teichteil-Königsbuch. Stochastic safest and shortest path problems. In *Proc. Assoc. for Adv. of Artificial Intelligence (AAAI)*, pages 1825–1831, 2012.

[Viappiani and Boutilier, 2009] Paolo Viappiani and Craig Boutilier. Regret-based optimal recommendation sets in conversational recommender systems. In *Proc. ACM Conf. on Recommender Systems*, pages 101–108, 2009.

[Weng and Zanuttini, 2013] Paul Weng and Bruno Zanuttini. Interactive value iteration for Markov decision processes with unknown rewards. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 2415–2421, 2013.

[Witwicki and Durfee, 2010] Stefan J Witwicki and Edmund H Durfee. Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In *Proc. Int. Conf. Auto. Planning and Scheduling (ICAPS)*, pages 185–192, 2010.

[Zhang *et al.*, 2017] Shun Zhang, Edmund Durfee, and Satinder Singh. Approximately-optimal queries for planning in reward-uncertain Markov decision processes. In *Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 339–347, 2017.