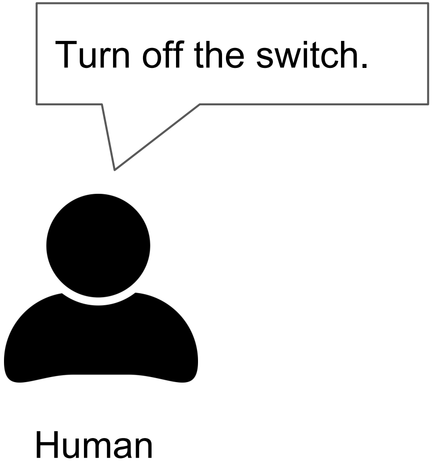
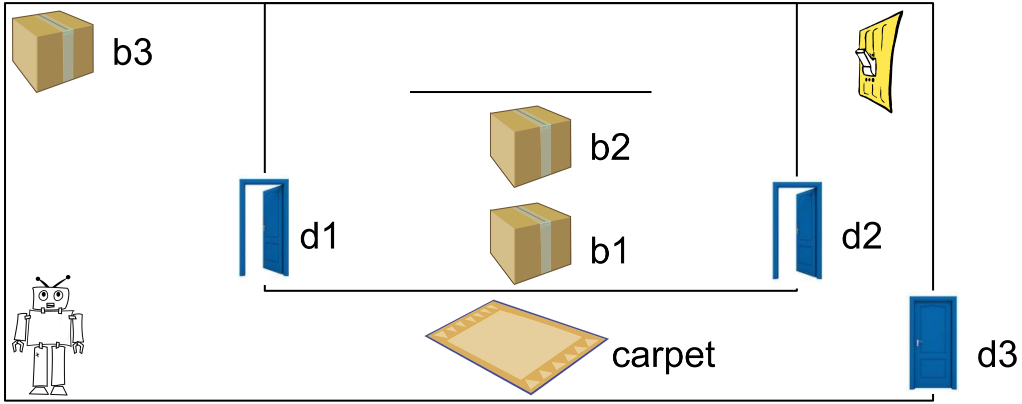
Minimax-Regret Querying on Side Effects for Safe Optimality in Factored Markov Decision Processes

Shun Zhang, Edmund H. Durfee, and Satinder Singh
University of Michigan

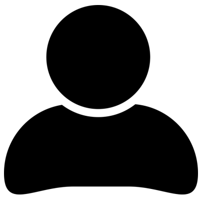
Motivating Example



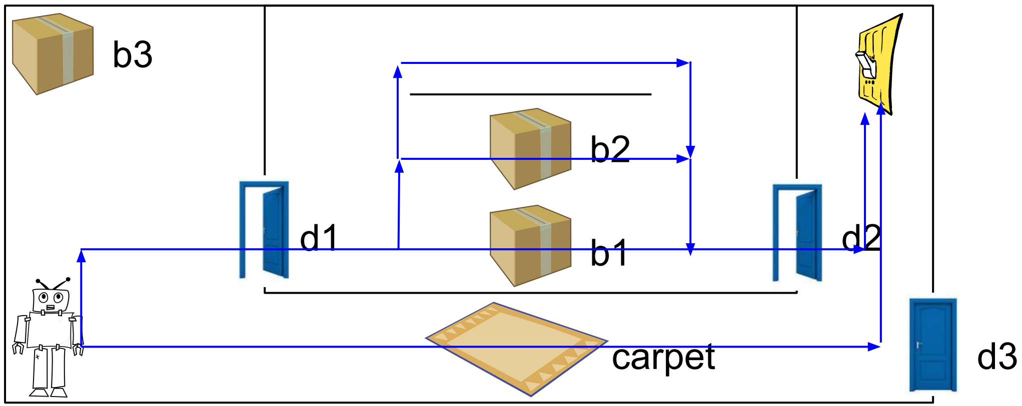


Motivating Example

Turn off the switch.

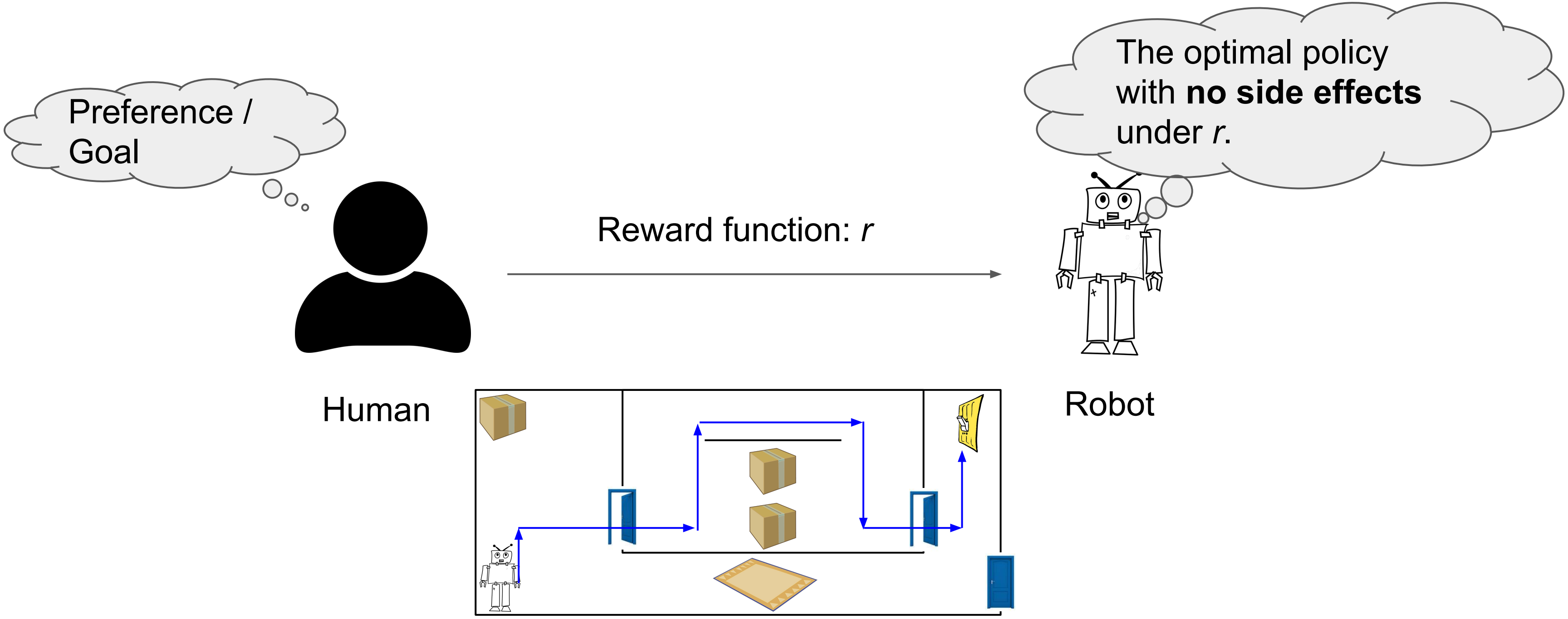


Human



Concerns: Policies often have **side effects**.

Baseline Performance



In this work, we allow the robot to **query** about the allowability of side effects.

Contributions

- We formulate the AI safety problem of **avoiding negative side-effects** in factored MDPs.
- If the robot can ask one query about if some side-effects are allowable, we show how to efficiently find a **minimax-regret query**.

Problem Formulation: Factored Representation

The domain is a factored MDP and known to the robot.
States are represented by a set of features $\{\phi_1, \phi_2, \ldots, \phi_n\}$

An example state:
(**RobotLocation** = intial_location,
Door1 = open,
Carpet = clean, …)

Problem Formulation: Factored Representation

The domain is a factored MDP and known to the robot.
States are represented by a set of features $\{\phi_1, \phi_2, ..., \phi_n\}$

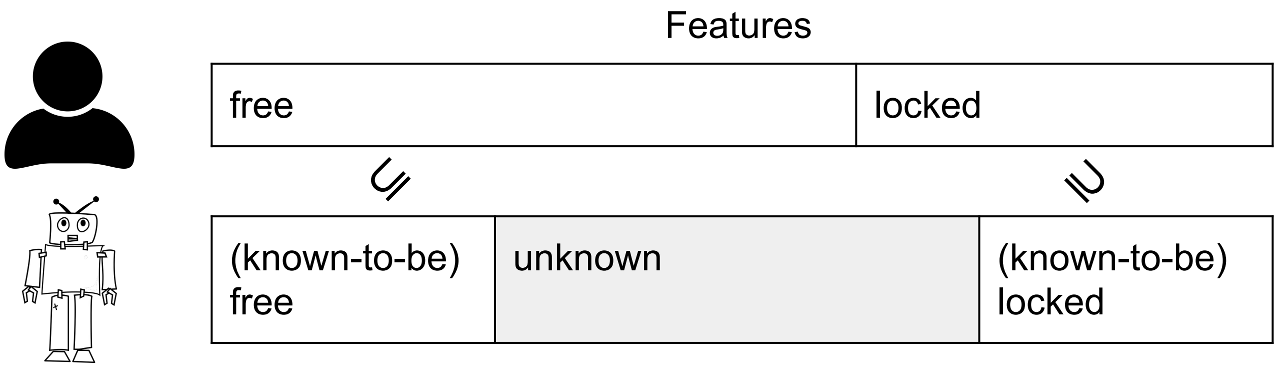
Features

free	locked

Negative side-effects arise when a **human's locked** feature is changed.

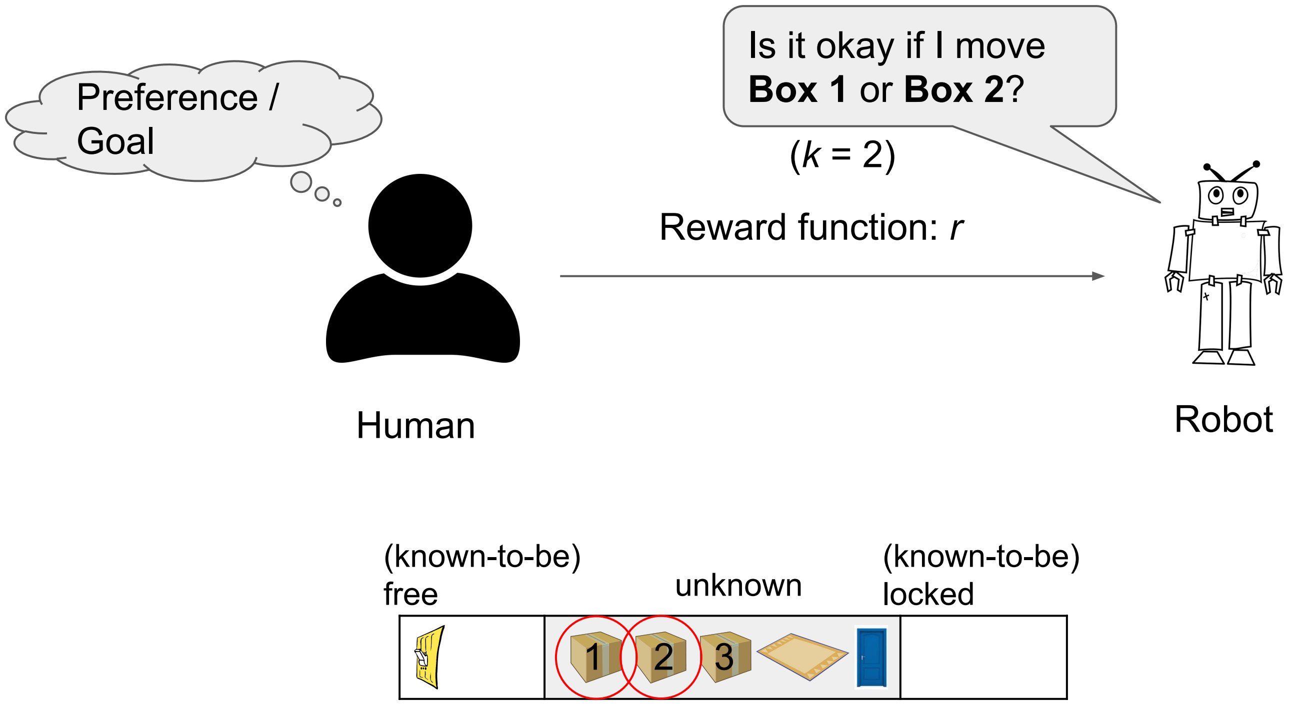
Problem Formulation: Factored Representation

The domain is a factored MDP and known to the robot.
States are represented by a set of features $\{\phi_1, \phi_2, \ldots, \phi_n\}$

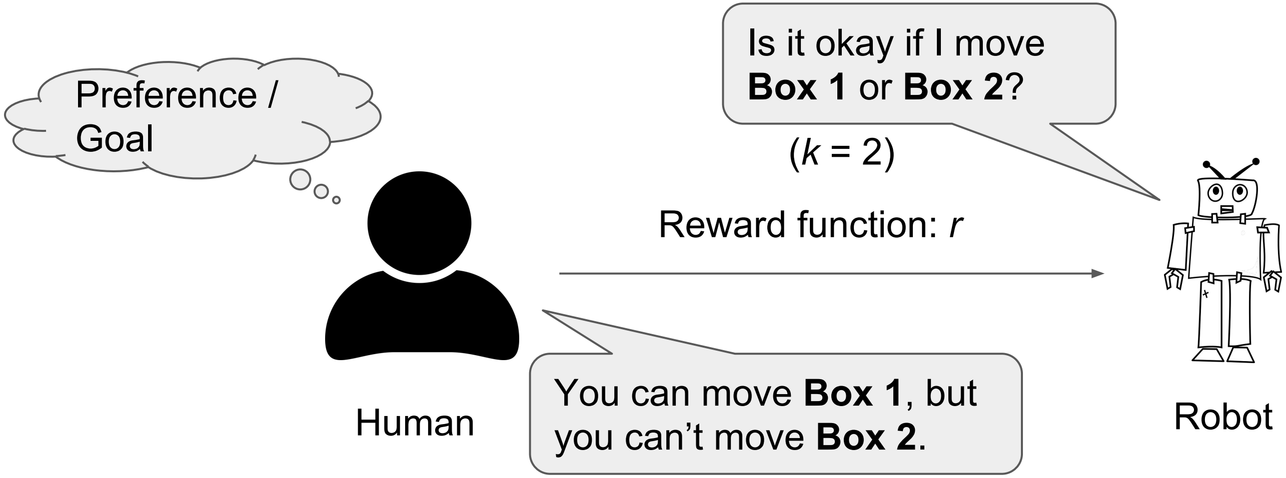


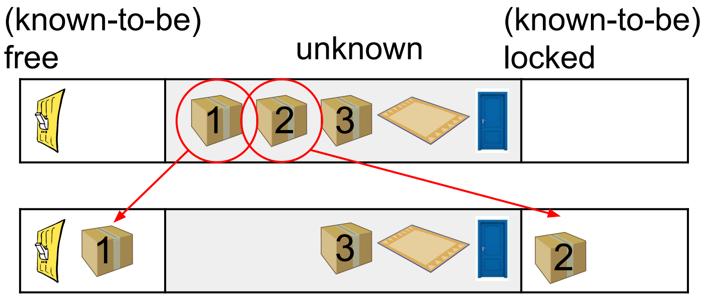
Negative side-effects arise when a **human's locked** feature is changed.
Safety constraints: The robot never changes any **known-to-be-locked** or **unknown** features.

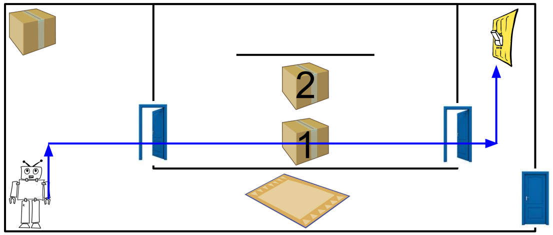
Problem Formulation: *k*-Feature Queries



Problem Formulation: *k*-Feature Queries

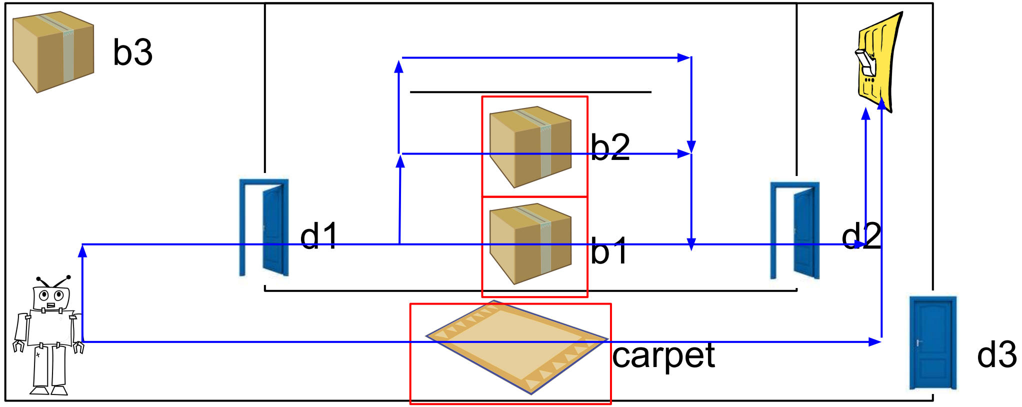






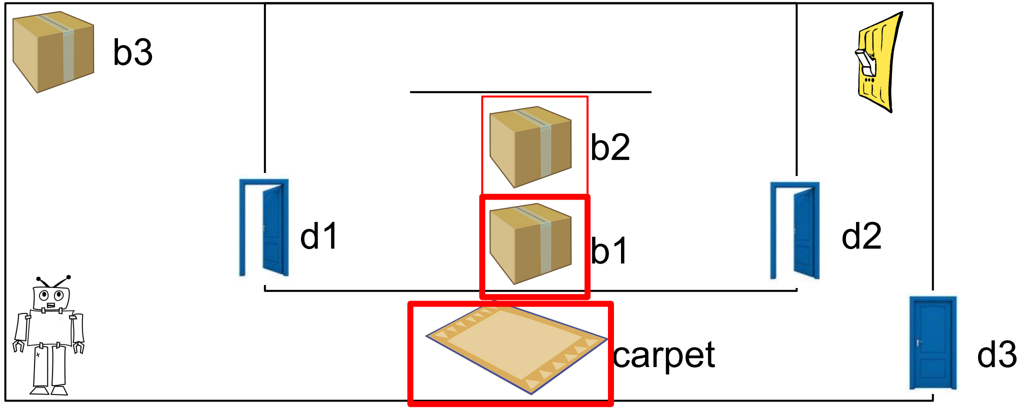
Method: A Two-Step Procedure

- **Find the relevant features. Efficiently and provably finding all relevant features (Algorithm *DomPis* in the paper).**
- Find an optimal *k*-feature query.



Method: A Two-Step Procedure

- Find the relevant features. Efficiently and provably finding all relevant features (Algorithm *DomPis* in the paper).
- **Find an optimal *k*-feature query.**



Example: k = 2

Minimax-Regret Queries

The objective is to find a query that has the **minimax regret**.

$V^*|q, \boxed{\text{locked}}\boxed{\text{free}}$

The value of the optimal policy if q is asked
and the partition of features in the human's mind
is $\boxed{\text{locked}}\boxed{\text{free}}$

Minimax-Regret Queries

The objective is to find a query that has the **minimax regret**.

$$\underbrace{\left(V^*|_{q',\,\boxed{\text{locked}}\boxed{\text{free}}} - V^*|_{q,\,\boxed{\text{locked}}\boxed{\text{free}}}\right)}_{\textbf{regret}\text{ of asking q instead of }q'\text{ under }\boxed{\text{locked}}\boxed{\text{free}}}$$

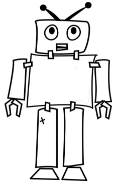
Minimax-Regret Queries

The objective is to find a query that has the **minimax regret**.

$$\underbrace{\max_{q',\ \boxed{\text{locked}}\boxed{\text{free}}} \underbrace{\left(V^*|q',\ \boxed{\text{locked}}\boxed{\text{free}} - V^*|q,\ \boxed{\text{locked}}\boxed{\text{free}}\right)}_{\textbf{regret}\text{ of asking q instead of q' under } \boxed{\text{locked}}\boxed{\text{free}}}}_{\textbf{maximum (worst-case) regret}\text{ of asking q}}$$

Minimax-Regret Queries

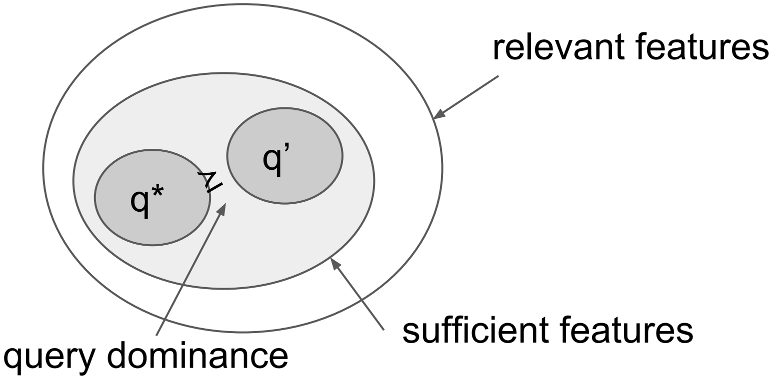
The objective is to find a query that has the **minimax regret**.



$$\underbrace{\min_{q} \underbrace{\max_{q',\ \boxed{\text{locked}}\boxed{\text{free}}} \underbrace{\left(V^*|q', \boxed{\text{locked}}\boxed{\text{free}} - V^*|q, \boxed{\text{locked}}\boxed{\text{free}}\right)}_{\textbf{regret} \text{ of asking q instead of q' under } \boxed{\text{locked}}\boxed{\text{free}}}}_{\textbf{maximum (worst-case) regret} \text{ of asking q}}}_{\textbf{minimax regret}}$$

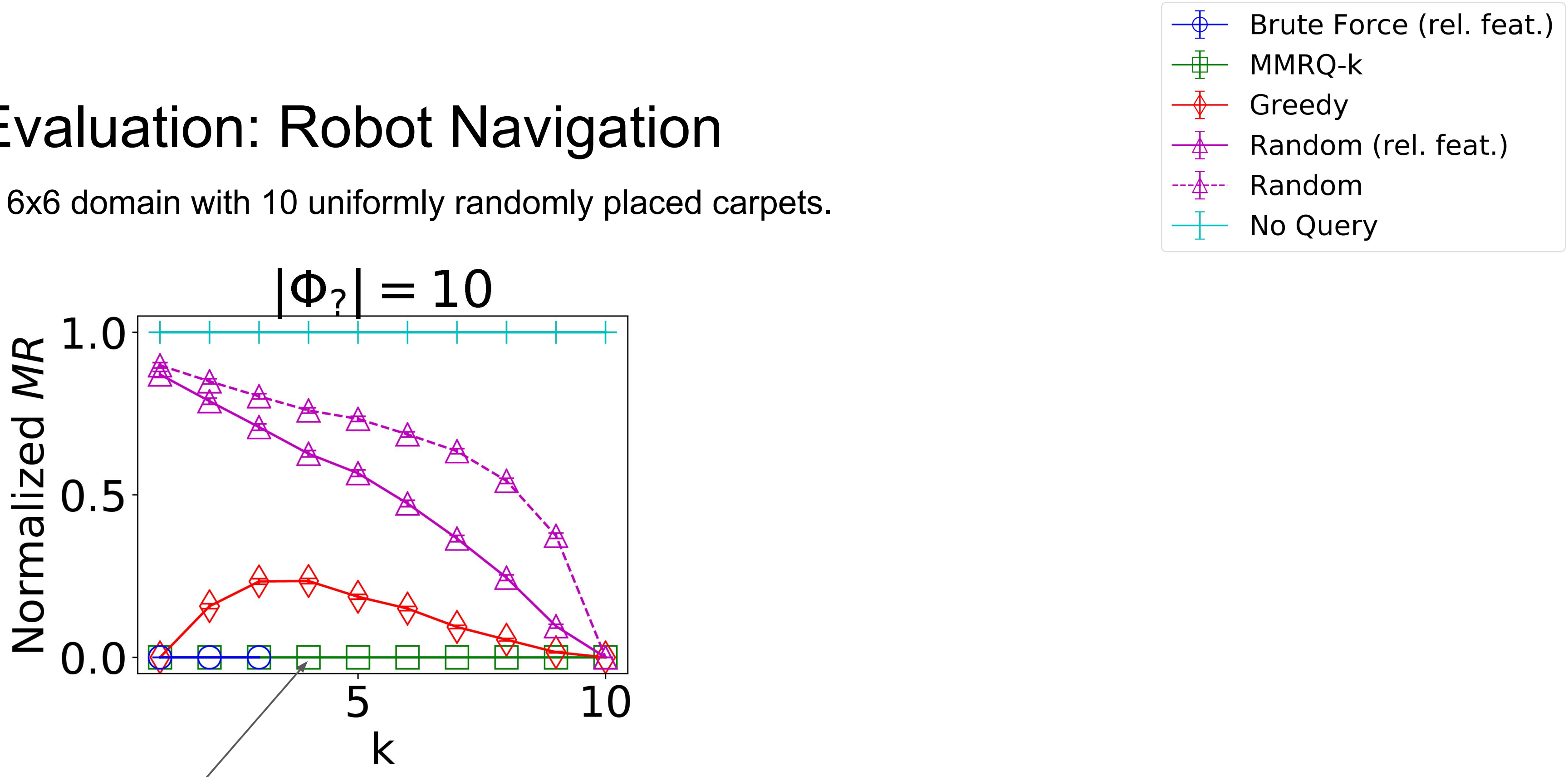
Find a Minimax-Regret *k*-Feature Query

Our algorithm ***MMRQ-k*** provably finds a minimax-regret query without evaluating all *k*-feature queries using the following techniques:
Early stopping when it finds a set of **sufficient features**.
Pruning some unevaluated queries by **query dominance**.
(More details in the paper.)



Evaluation: Robot Navigation

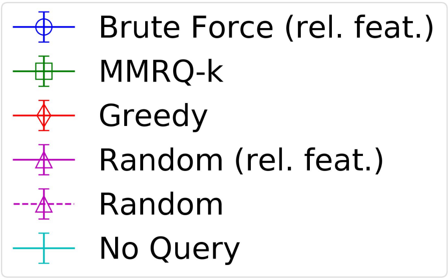
A 6x6 domain with 10 uniformly randomly placed carpets.

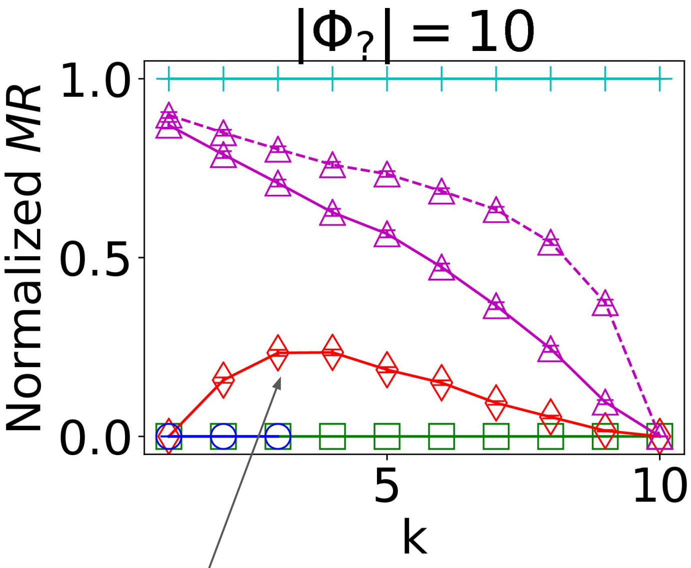


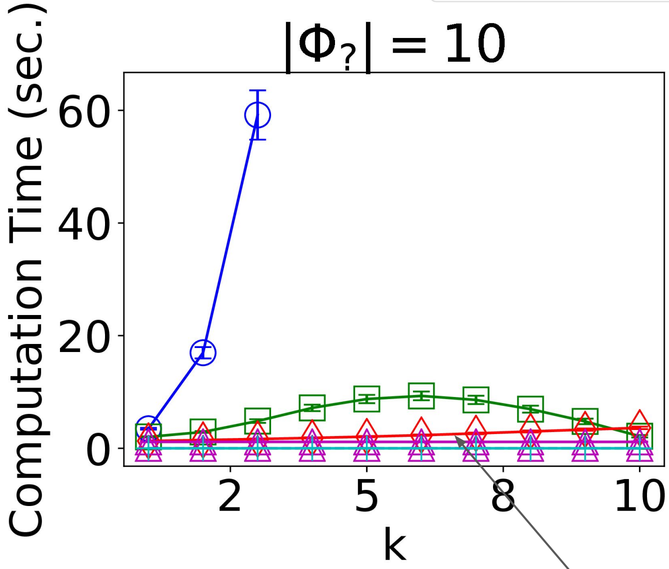
MMRQ-k always finds the minimax-regret query.

Evaluation: Robot Navigation

A 6x6 domain with 10 uniformly randomly placed carpets.





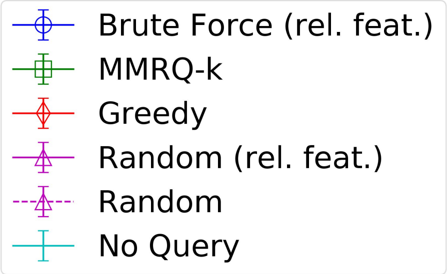


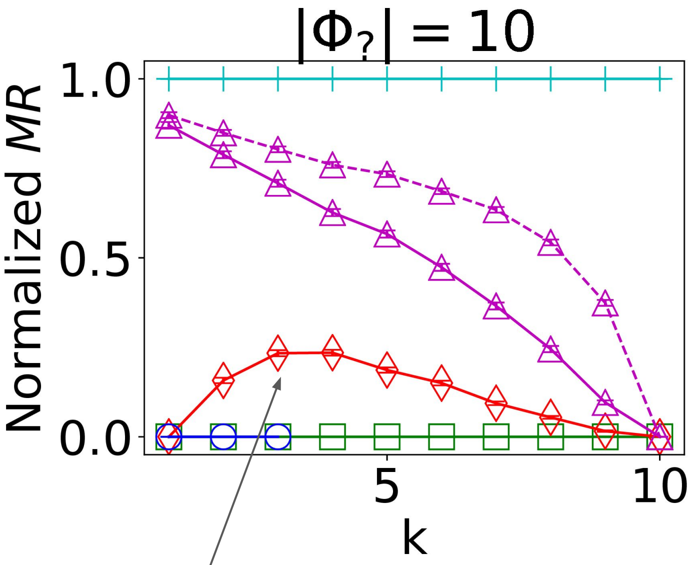
The greedy approach has worse performance compared with MMRQ-k.

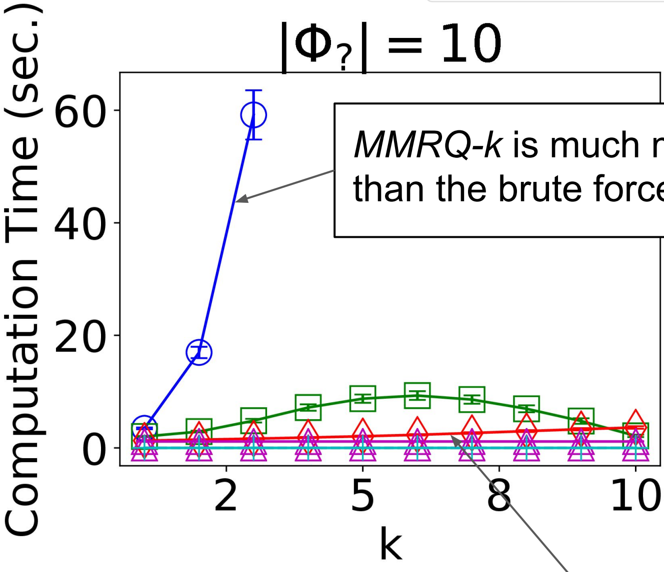
MMRQ-k is less efficient than the greedy approach.

Evaluation: Robot Navigation

A 6x6 domain with 10 uniformly randomly placed carpets.







MMRQ-k is much more efficient than the brute force method.

The greedy approach has worse performance compared with MMRQ-k.

MMRQ-k is less efficient than the greedy approach.

Summary

- Formulation of avoiding negative side-effects in factored MDPs.
- An algorithm that provably and efficiently finds relevant features.
- An algorithm that provably and efficiently finds minimax-regret queries given a limit on the number of features the robot can ask about.
- Future work: approximate methods; other changeability assumptions.

Backup Slides

Safely-Optimal Policy

A safely-optimal policy is the optimal policy that does not change locked features or unknown features. It can be found by solving a linear programming problem.

occupancy frequency on state, action pairs

1 if s' = s_0; 0 otherwise

$$\max_{x} \sum_{s,a} x(s,a) r(s,a)$$

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

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

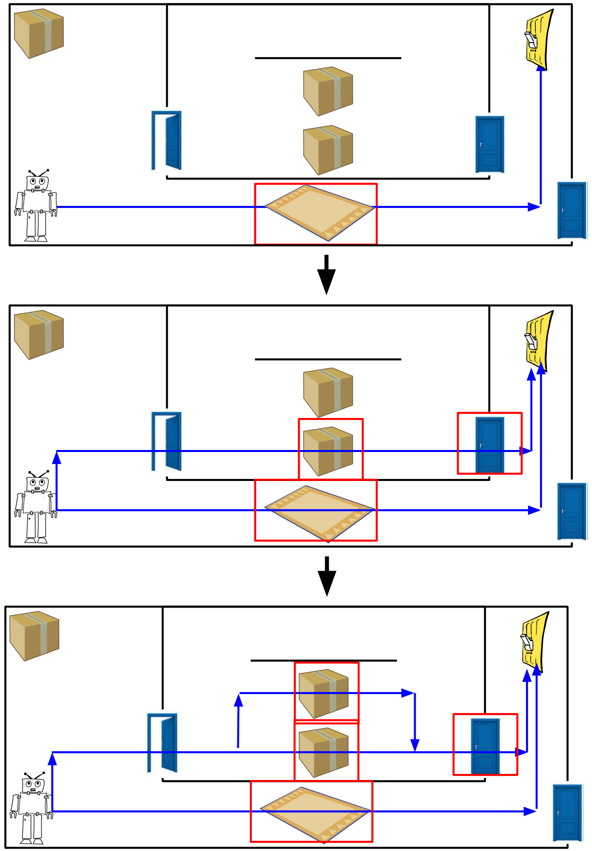
States with changed locked features or unknown features should not be visited.

Construction of the Set of Dominating Policies

We use a greedy construction method to find all relevant features.

Correctness: Our algorithm finds all relevant features (and only relevant features).

Efficiency: Computation time is exponential in the number of relevant features in the worst case.
The paper specifies a pruning rule to avoid considering all subsets of relevant features.



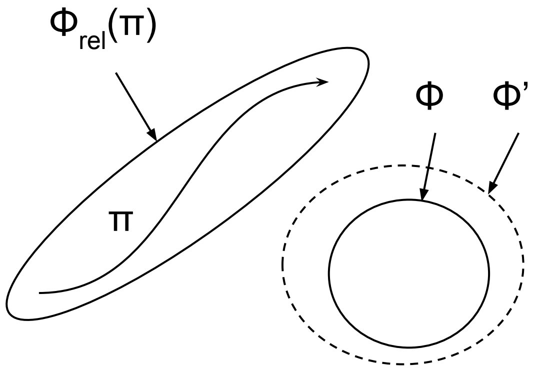
Construction of the Set of Dominating Policies: Algorithm

Algorithm *DomPolicies*

$\Gamma' \leftarrow \emptyset$ ▹ the initial set of dominating policies
$\Phi'_{rel} \leftarrow \emptyset$ ▹ the initial set of relevant features
$checked \leftarrow \emptyset$ ▹ It contains $\Phi \subseteq \Phi'_{rel}$ we have examined so far.
$\beta \leftarrow \emptyset$ ▹ a pruning rule
$agenda \leftarrow powerset(\Phi'_{rel}) \setminus checked$
while $agenda \neq \emptyset$ **do**
  $\Phi \leftarrow$ an element with the smallest cardinality from *agenda*
  if $satisfy(\Phi, \beta)$ **then**
    (find the safely-optimal policy that does not change Φ)
    $\pi \leftarrow \arg\max_{\pi' \in \Pi_\Phi} V^{\pi'}$, ▹ by solving Eq. 1
    if π exists **then**
      $\Gamma' \leftarrow \Gamma' \cup \{\pi\}$
      add $(\Phi, \Phi_{rel}(\pi))$ to β
    end if
  end if
  $\Phi'_{rel} \leftarrow \Phi'_{rel} \cup \Phi_{rel}(\pi)$
  $agenda \leftarrow powerset(\Phi'_{rel}) \setminus checked$
  $checked \leftarrow checked \cup \{\Phi\}$
end while
return Γ'

Correctness: guaranteed to find all relevant features.

Computational efficiency: exponential in the number of relevant features in the worst case. The efficiency can be improved with pruning:



Defined similar to Regan and Boutilier (2010)

Minimax-Regret k-Feature Queries

The **utility** of a *k*-feature query Φ_q when Φ_C are the changeable features.

$$u(\Phi_q, \Phi_C) = \max_{\pi \in \Pi_{\Phi_? \setminus \{\Phi_q \cap \Phi_C\}}} V^\pi$$

The **pairwise maximum regret** of asking Φ_q rather than $\Phi_{q'}$.

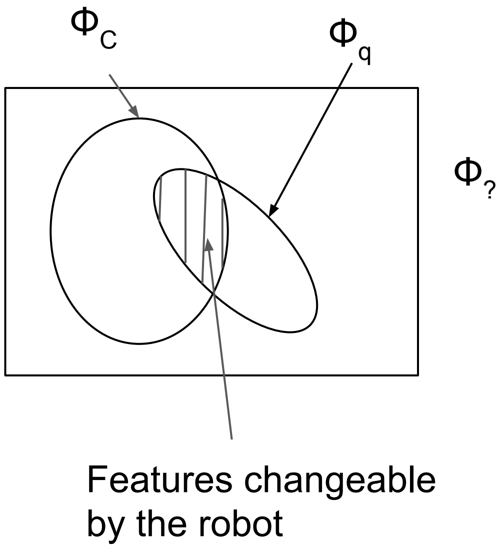
$$PMR(\Phi_q, \Phi_{q'}) = \max_{\Phi_C \subseteq \Phi_?} (u(\Phi_{q'}, \Phi_C) - u(\Phi_q, \Phi_C))$$

The **maximum regret** of asking Φ_q.

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

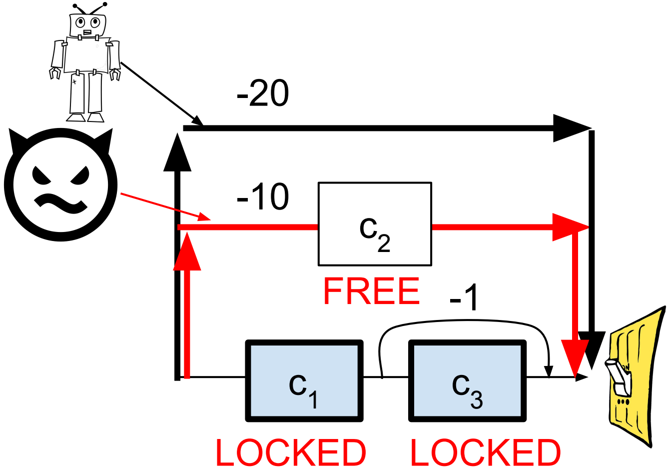
The **minimax-regret** *k*-feature query.

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

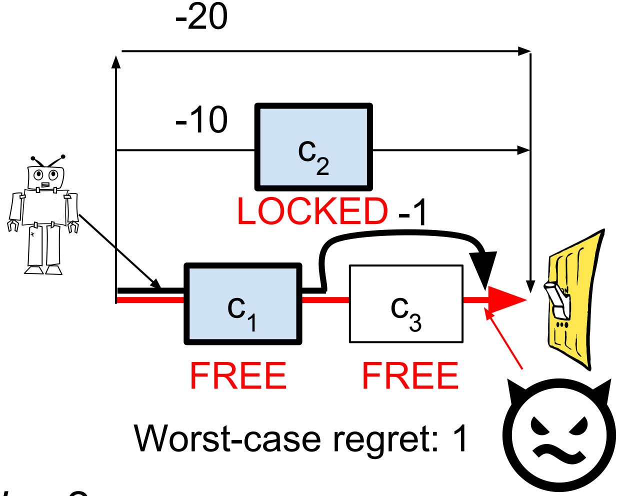


Minimax-Regret Queries

The objective is to find a query that has the **minimax regret**.
A greedy approach based on a prior work (Viappiani and Boutilier, 2009) may not always find a minimax-regret query.



Worst-case regret: 10



Worst-case regret: 1

Example: k = 2.

Motivated by Viappiani and Boutilier (2009).

Chain of Adversaries

An efficient greedy construction algorithm that is linear in *k*.

It may not find the minimax-regret query.

Algorithm ***CoA*****: Chain of adversaries**

Initialize $\Phi_{q_0} \leftarrow \emptyset$, $i \leftarrow 0$.
while $|\Phi_{q_i}| < k$ and ($i = 0$ or $\Phi_{q_{i-1}} \neq \Phi_{q_i}$) **do**
    $k' \leftarrow k - |\Phi_{q_i}|$
    $\Phi_{q_{i+1}} \leftarrow \Phi_{q_i} \cup \boxed{\Phi_{rel}(\pi^{MR_{k'}}_{\Phi_{q_i}})}$
    $i \leftarrow i + 1$
end while
return Φ_{q_i}

Consider what features an adversary wants to change when we asks our current query, and add those features to our query.

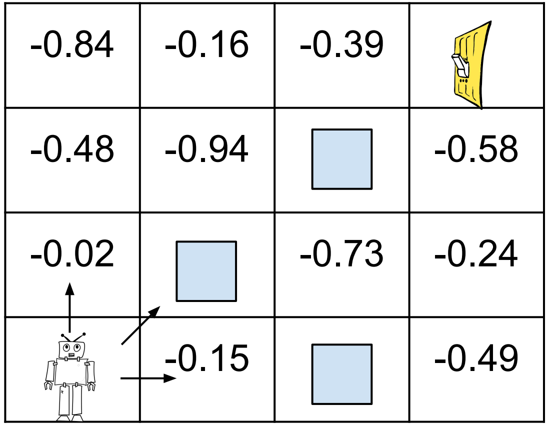
Office Navigation: Domain Description

The robot is tasked to turn off a switch at a corner.

The domain is 6x6 with 10 uniformly randomly placed carpets.

The reward is 0 to for any location with a carpet, and uniformly random in [-1, 0] for any location without a carpet. So the robot prefers to walk on the carpets.

Each carpet corresponds to one unknown feature. The robot initially does not know if it can traverse any carpet.



(Illustration in a reduced size)

More Empirical Results