Efficiently Finding Approximately-Optimal Queries for Improving Policies and Guaranteeing Safety

by

Shun Zhang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2020

Doctoral Committee:

Professor Satinder Singh Baveja, Co-Chair
Professor Edmund H. Durfee, Co-Chair
Professor Walter S. Lasecki
Professor Richard L. Lewis

Shun Zhang
shunzh@umich.edu
ORCID iD: 0000-0001-5046-6579

ACKNOWLEDGMENTS

First and foremost I would like to thank my co-advisors, Satinder Singh and Ed Durfee, for their joint efforts in training me as independent researcher. They encouraged me to identify the fundamental problems and contribute to providing provably-correct solutions. I also appreciate Ed's efforts and patience in providing detailed feedback to my weekly reports and paper drafts. Ed gave me valuable feedback on both of the research itself and how to present the research, and inspired me to find better ways to present my research. The research in this dissertation would not be possible without their support.

I would also like to thank my committee members, Walter Lasecki and Richard Lewis, for their insightful feedback on my proposal and my defense. They provided useful feedback on what was missing to complete the dissertation. Also, my assumption on the human's response model was quite simplistic. They challenged the formulation of the human-agent interaction process from a useability perspective. I was able to make my dissertation stronger by addressing these issues.

Lastly, I would like to thank my colleagues and the CSE department for their support. I would like to thank Rob Cohn for his advice on my research in my first year. He was always available to answer my questions on this research topic and share his experiences and thoughts. I would also like to thank other current and former labmates for their support and useful conversations: Qi Zhang, Xiaoxiao Guo, Nan Jiang, Junhyuk Oh, Janarthanan Rajendran, Aditya Modi, Christopher Grimm, Zeyu Zheng, Vivek Veeriah, Ethan Brooks, and Wilka Carvalho.

TABLE OF CONTENTS

ACKNOWLEDGMENTS ii

LIST OF FIGURES vi

ABSTRACT viii

CHAPTER

1. Introduction 1

1.1 Problem Statement 2
1.2 Contributions 5
1.3 Approaches 6

2. Background 8

2.1 Markov Decision Processes 8
2.1.1 Factored MDPs 9
2.1.2 Reward-Uncertain MDPs 10
2.1.3 Constrained MDPs 11
2.2 Query Selection Problem 12
2.3 Combinatorial Optimization and Submodularity 13

3. Problem Statement 15

3.1 Efficiently Finding Approximately-Optimal Queries Under (Only) Reward Uncertainty 16
3.2 Efficiently Finding Optimal or Empirically Good Queries Under (Only) Safety-Constraint Uncertainty 17
3.2.1 Improving a Safe Policy 19
3.2.2 Finding an Initial Safe Policy 19
3.3 Low-Cost Querying Under Both Reward and Safety-Constraint Uncertainty 19
3.4 Discussion on the Querying Semantics 20

4. Related Work . . . 22

4.1 Human-Agent Interaction and Preference Elicitation . . . 22
4.2 Reward Design . . . 24
4.3 AI Safety . . . 25

5. Efficiently Finding Approximately-Optimal Queries Under (Only) Reward Uncertainty . . . 27

5.1 Expected Posterior Utility and Expected Utility of Selection . . . 28
5.2 Finding Policy Queries . . . 29
5.2.1 Greedy Construction of Policy Queries . . . 31
5.3 Finding Trajectory Queries by Projection . . . 34
5.4 Empirical Evaluations . . . 36
5.4.1 Comparison with a Sampling Algorithm . . . 37
5.4.2 Comparison with the Optimal Query . . . 38
5.4.3 Evaluation of Trajectory Queries . . . 39
5.4.4 Evaluation in Discrete Driving Domain . . . 42
5.5 Conclusion . . . 44

6. Efficiently Finding Optimal Queries Under (Only) Safety-Constraint Uncertainty . . . 47

6.1 Problem Definition . . . 48
6.1.1 Finding Safely-Optimal Policies . . . 49
6.2 Finding All Relevant Unknown-Features . . . 50
6.3 Finding Minimax-Regret Queries . . . 54
6.4 Empirical Evaluations . . . 59
6.4.1 Robot Navigation . . . 60
6.5 Discussion and Conclusion . . . 62

7. Querying to Find an Initial Safe Policy . . . 65

7.1 Problem Statement . . . 66
7.2 Querying to Find a Safe Policy . . . 68
7.2.1 Myopic Query Selection . . . 68
7.2.2 Set Cover Formulation . . . 70
7.2.3 Set-Cover-Based Algorithm . . . 72
7.3 Empirical Evaluation . . . 77
7.4 Conclusion . . . 82

8. Query Selection Under Joint Uncertainty . . . 83

8.1 Problem Statement . . . 83
8.2 Query Selection Methods . . . 84

8.2.1 Myopic Heuristic 85
8.2.2 Dominating-Policy-Based Heuristic 87
8.2.3 Batch-Query-Based Heuristic 88
8.3 Empirical Evaluations 90
8.4 Implementation Details: The MILP Formulation 93
8.5 Conclusion 97

9. Conclusion 99

9.1 Summary of Contributions 99
9.2 Future Work 100
9.3 Summary 102

BIBLIOGRAPHY 103

LIST OF FIGURES

Figure

5.1 Greedy policy construction illustration. 33
5.2 Reward settings of the rock collection domain. The locations of rocks are randomly assigned. 37
5.3 Comparison of methods for finding policy queries in the rock collection domain. $n = 10, k = 2$. 38
5.4 Legend for Figures 5.5 and 5.6. 40
5.5 $EVOI$ of policy query selection methods in the rock collection domain. Legend is in Figure 5.4. 40
5.6 Computational time of policy query selection methods in the rock collection domain. Legend is in Figure 5.4. 41
5.7 Legend for Figures 5.8, 5.9 and 5.11. 43
5.8 Comparison in the rock collection domain. Legend is in Figure 5.7. 43
5.9 Computational time in the rock collection domain. Legend is in Figure 5.7. 44
5.10 The driving domain. 45
5.11 Evaluation on the driving domain. The horizontal axis is the number of trajectories in the trajectory query (k). The optimal query q^*_Π cannot be computed. 46
6.1 The robot navigation domain. The dominating policies are shown as arrows. 49
6.2 Pruning rule illustration. 53
6.3 Example domains used in text. ($n > 2$) 54
6.4 Theorem 6.1 illustration. 54
6.5 (Left) Illustration of the set of features the robot can change, indicated by the shaded area. (Right) Illustration of the proof of Theorem 6.2. 57
6.6 Example domain illustrating that *CoA* does not always find the minimax-regret query. 60
6.7 Office navigation and legend for following figures. 62
6.8 The legend for the following figures. 63
6.9 Normalized maximum MR vs. k. $|\Phi_?| = 10$. Brute force computation time is only shown for $k = 0, 1, 2$. 63
6.10 Normalized MR vs. the number of relevant features. $|\Phi_?| = 10$ and $k = 2, 4$. 63
7.1 The robot navigation domain. 65

7.2 Example with 4 unknown features (3 of which are relevant). In (a) I show relevant features of dominating policies; In (b) I show IISs. In (c) is an optimal query policy for the setting in Example 7.1. 'F' means the queried feature is free and 'L' means the queried feature is locked. 69
7.3 The robot navigation domain. The blue-colored tiles are carpets and dark tiles are walls. The dashed-line policy is safe if the robot knows that the traversed carpets are free to change. 78
7.4 Legend for the following figures. 79
7.5 (Left) The number of queried features vs. the number of unknown features (carpets). (Right) Computation time per query vs. the number of carpets. 79
7.6 The number of queried features vs. p_F. The horizontal axis is the midpoint of the intervals where p_F is sampled from, namely, $[0, 0.5], [0.1, 0.6], \ldots$. To see the differences clearly, I report a subset of algorithms in each figure. 79
7.7 Randomly placed 40 carpets and 20 walls in a 10×10 domain. Optimal is intractable to run and is not shown. 80
8.1 An example where the myopic heuristic finds a suboptimal query policy. . 86
8.2 An example where the dominating-policy-based heuristic finds a suboptimal query policy. 88
8.3 The experiment domain in this chapter. 91
8.4 An example where the batch-query-based heuristic does not find the optimal query policy. 92
8.5 An example where the batch-query-based heuristic finds a worse query policy than batch-query-based heuristic and myopic heuristic. 93
8.6 The legend for the following figures. 94
8.7 Empirical results on domains with 2 switches 94
8.8 Empirical results on domains with 4 switches 95
8.9 (Top) The differences of the objective values between the batch-query-based queries and the others in domains with 2 switches and 12 carpets. The vertical black segment indicates the number of trials where the difference between the two algorithms is 0. 96

ABSTRACT

When a computational agent (called the "robot") takes actions on behalf of a human user, it may be uncertain about the human's preferences. The human may initially specify her preferences incompletely or inaccurately. In this case, the robot's performance may be unsatisfactory or even cause negative side effects to the environment. There are approaches in the literature that may solve this problem. For example, the human can provide some demonstrations which clarify the robot's uncertainty. The human may give real-time feedback to the robot's behavior, or monitor the robot and stop the robot when it may perform anything dangerous. However, these methods typically require much of the human's attention. Alternatively, the robot may estimate the human's true preferences using the specified preferences, but this is error-prone and requires making assumptions on how the human specifies her preferences.

In this thesis, I consider a *querying* approach. Before taking any actions, the robot has a chance to query the human about her preferences. For example, the robot may query the human about which trajectory in a set of trajectories she likes the most, or whether the human cares about some side effects to the domain. After the human responds to the query, the robot expects to improve its performance and/or guarantee that its behavior is considered safe by the human.

If we do not impose any constraint on the number of queries the robot can pose, the robot may keep posing queries until it is absolutely certain about the human's preferences. This may consume too much of the human's cognitive load. The information obtained in the responses to some of the queries may only marginally improve the robot's performance, which is not worth the human's attention at all. So in the problems considered in this thesis, I constrain the number of queries that the robot can pose, or associate each query with a cost. The research question is how to efficiently find the most useful query under such constraints.

Finding a provably optimal query can be challenging since it is usually a combinatorial optimization problem. In this thesis, I contribute to providing efficient query selection algorithms under uncertainty. I first formulate the robot's uncertainty as *reward uncertainty* and *safety-constraint uncertainty*. Under only reward uncertainty, I provide a query selection al-

gorithm that finds approximately-optimal k-response queries. Under only safety-constraint uncertainty, I provide a query selection algorithm that finds an optimal k-element query to improve a known safe policy, and an algorithm that uses a set-cover-based query selection strategy to find an initial safe policy. Under both types of uncertainty simultaneously, I provide a batch-query-based querying method that empirically outperforms other baseline querying methods.

CHAPTER 1

Introduction

In artificial intelligence, an autonomous agent can achieve super-human performance in some well-defined domains. We have seen many successes in recent years, including Atari games (Mnih et al. 2013) and the game of Go (Silver et al. 2016). Given its computational power, an autonomous agent may find a policy that pleasantly surprises us. One example is move 37 in the second game between AlphaGo and Lee Sedol (Silver et al. 2016). It is a move by AlphaGo that surprises even professional Go players. The human Go players only realized after the game that such a move is novel and helped AlphaGo win the game.

However, there is also a chance that an autonomous agent would *negatively* surprise us. In most real-world problems, our preferences may not be completely or correctly specified. The autonomous agent may optimize an uncertain or incorrect objective. In this case, the agent's performance may be unsatisfactory or its behavior can cause unsafe side effects to the environment.

One popular example of negative surprise is the racing-boat problem (Clark and Amodei 2016). There are bonuses (positive rewards) that the boat can collect along a racing track. The boat is expected to reach the goal as fast as possible by collecting the bonuses. However, the game is designed in a way such that bonuses can reappear after the boat collects them. The boat ends up with learning a policy that circulates around some bonuses and endlessly collects them, without ever reaching the goal.[1] Another more real-world example is video recommendation (Chaslot 2019). A video website wants to maximize the length of time that a user is engaged with the website. If we train a video recommendation system that maximizes the length of time of user engagement, we expect that it can find videos that match the user's preferences. However, in reality, misinformation and rumors can draw much more attention than other types of videos. Although our intention is to recommend videos that match the users' interests, the recommendation

[1] This example is illustrated in this video (`https://youtu.be/tlOIHko8ySg`).

system may find recommending videos of misinformation and rumors easily increases the users' engagement. Such a strategy is not what we expect.

This issue is gaining more attention in the literature in recent years. It is referred to as human-agent value alignment, reward hacking, or avoiding negative side effects (Amodei et al. 2016; Leike et al. 2017). This thesis lies in this literature and is motivated by the fact that a human user may not be able to predefine the correct objective that an autonomous agent should optimize. To resolve the agent's uncertainty, I adopt a *querying* approach. Whenever the agent is uncertain about the objective, it has a chance to proactively query. After querying, the agent expects to be more certain about the human's preferences and guarantees that its policy would not negatively surprise the human.

Although we can define some straightforward heuristics that help to find a good query, finding a provably (approximately-)optimal query in some settings can be extremely challenging. I use an accepted criterion in the literature to evaluate queries, called *expected value of information* (EVOI) (Cohn, Singh, and Durfee 2014; Viappiani and Boutilier 2010). Unfortunately, finding the query that maximizes such an objective is usually a combinatorial optimization problem, which can be computationally expensive. In this thesis, I contribute to both formulating precise querying under uncertainty problems and providing query selection algorithms that either find a provably (approximately-)optimal query or find a query that is empirically significantly better than state-of-the-art methods.

1.1 Problem Statement

I briefly describe the scope of problems considered in this thesis. I will define the problems more formally and in detail in Ch. 3.

I consider a setting where an autonomous agent (called the *robot*) takes actions on behalf of a human user (called the *human*). The human may not have the capacity or patience to specify her preferences completely. Since I formulate the domain as a Markov decision process (which will be formally defined in Ch. 2), the human's preferences are captured by a *reward function*. If the robot ignores such uncertainty about the human's preferences and simply optimizes the incompletely-specified reward function, its behavior may be unsatisfactory or even negatively surprise the human. I formulate the robot's uncertainty in the following two forms: *reward uncertainty* and *safety-constraint uncertainty*, The robot can only resolve its uncertainty by *querying* the human user. I first introduce the two types of uncertainty.

Reward uncertainty. Instead of knowing the exact true reward function, the robot knows a set of possible reward functions that are consistent with the human's preference specification. Only one of the reward functions in this set is the true reward function in the human's mind. The robot's policy may not be satisfactory because it does not know which reward function to optimize. Given a prior belief over the possible reward functions, the robot could optimize the mean reward function, but the corresponding optimal policy may not be satisfactory under the true reward function. I call this form of uncertainty reward uncertainty.

The robot can query to reduce its reward uncertainty to improve its performance. For example, it could ask which reward function is the true reward function, or which subsets of reward functions contain the true reward function. From a useability perspective, it may be difficult to communicate reward functions directly. So the robot can provide a set of trajectories and ask the human to pick the one that has the highest value. I will formulate this problem in Sec. 3.1.

Example 1.1. *We consider a domestic robot example. Suppose a human user asks a robot to get her a drink. She may have had tea or coffee in the past and does not specify what she wants for today. The robot does not know which drink to bring if it can only bring one drink. In this case, the robot has reward uncertainty.*

Under such uncertainty, the robot can provide a visualization of two trajectories where one of them makes coffee and the other one makes tea. The human responds with which trajectory she prefers. After the response, the robot is more certain about the true reward function and can improve its policy.

Safety-constraint uncertainty. The robot's policy may be *unsafe* and negatively surprise the human because of the incomplete reward function. This problem is considered as *negative side effects* in the AI safety literature (Amodei et al. 2016). Indeed, if the robot can resolve the reward uncertainty and recover the true reward function, it would not negatively surprise the human. However, in reality, the robot may never recover the true reward function. In this case, I assume that the human has a set of safety constraints that the robot needs to follow. *The robot should guarantee that its behavior is safe by not violating any of these safety constraints*. The robot may be uncertain about what safety constraints it needs to follow. I call this form of uncertainty safety-constraint uncertainty.

Without any knowledge about what behaviors are safe, the robot may find it impractical to find a guaranteed-to-be-safe policy. It may end up with querying about whether all the states it possibly could visit are safe. So I assume that the robot knows a set of possible safety constraints that the human may have (for example, the vase should not be broken;

the box should not be moved, etc.). It can only ignore a possible safety constraint when it queries the human and confirms that the human does not require that safety constraint to hold (for example, the human may not mind if the robot moves the box). I will formulate this problem in Sec. 3.2.

Example 1.2. *In the domestic robot example, there could be other objects in the room (carpets, boxes, etc.) that the human did not mention in the designed reward function. The human only specifies a positive reward for getting a drink, but does not specify rewards for these objects. The robot may get a drink faster by traversing a carpet (which makes it dirty), moving a box away, or scaring away the user's cat. It may be uncertain about whether such side effects to the environment are allowed by the human (even though they are not reflected in the reward function). The robot may have uncertainty about which safety constraints it needs to follow.*

The robot may initially find that there is no known-to-be-safe path to get the drink. For example, the robot has to traverse a carpet to get to the drink, while it is uncertain about if it can make the carpet dirty or not. So the robot can query about if such a side effect is allowable. It will only execute a policy that traverses the carpet if the human confirms that it is allowed to do so. Otherwise, it will inform the human that there is no safe way to get the drink. Similarly, when the robot finds a safe policy, it can also query about other possible constraint relaxations which would improve its policy.

Also, I do not adopt the simple approach of formulating safety-constraint uncertainty as reward uncertainty (we could discourage unsafe behaviors using negative rewards). Since the robot guarantees to not violate any safety constraints, it is hard to provide such guarantees when constraints are represented as rewards: The robot's policy may still visit states with possible negative rewards. I will compare my work with works formulating unsafe behaviors as negative rewards in the related work chapter (Sec. 4.3).

Query selection problem. Under reward and/or safety-constraint uncertainty and without communication with the human, the robot's behavior may be unsatisfactory or unsafe. So I allow the robot to query the human to improve its performance. There are certainly other ways to communicate. One possible way explored in the literature is to let the human demonstrate a (close-to-)optimal policy or keep supervising the robot to provide feedback. Although these methods have successful applications in some human-robot interaction tasks (Knox and Stone 2009; Torrey and Taylor 2013), they require significant effort from the human. Additionally, the human may be uncertain about what uncertainty the robot has and what affects its planning the most: She may provide information which she

believes would help reduce the robot's uncertainty, but the robot does not find that helpful. So in this thesis, I assume that the human does not have the patience or ability to supervise the robot throughout its execution, or the insight to provide useful information. Instead, *the robot takes the initiative to query about the human's preference to improve its policy and/or to guarantee that its behavior is safe.*

If the robot queries about everything it is uncertain about, it would be tedious for the human to answer. So we want to impose a budget on the size of the query, or associate each query with a cost. Then it becomes crucial to find out what is most useful to ask about. Finding an optimal query can be challenging. In the following section, I describe how this thesis contributes to finding provably (approximately-)optimal or empirically good queries.

I should emphasize that, in this thesis, a query is a mathematical representation of the information that should be asked. Natural language processing and a human-computer interface will often be needed to realize such queries (Goodrich and Schultz 2007; Thomason et al. 2015), but these are beyond the scope of this thesis.

1.2 Contributions

This thesis makes the following contributions to the literature. I do not claim that this thesis solves the human-agent value alignment problem. Instead, it contributes to providing provably-correct or empirically-good solutions to the sub-problems of query selection under two forms of preference uncertainty: reward uncertainty and safety-constraint uncertainty, as summarized in Table 1.1.

Contribution I. Under only reward uncertainty, I provide an algorithm that finds an approximately-optimal query that is more efficient than the state-of-the-art algorithms (Ch. 5). The problem of selecting queries to resolve reward uncertainty is well explored in the literature (as we will see in the related works in Ch. 4). Most algorithms in the literature are based on heuristics. They may be justifiable choices for certain domains and have good empirical performance, but they do not provide an optimality guarantee. My contribution focuses on providing an algorithm that has an optimality guarantee by exploiting submodularity, policy generation using mixed integer linear programming, and a projection approach. To the best of our knowledge, this algorithm is the first querying algorithm that provides an optimality guarantee for sequential decision making problems.

Contribution II. Under only safety-constraint uncertainty, there are two possible cases: either the robot does not initially know a safe policy (that does not violate any safety con-

straints) or it initially knows a safe policy. When the robot initially knows a safe policy, I provide an algorithm that uses a pruning technique to find an optimal query that finds a better safe policy (Ch. 6). When the robot initially does not know a safe policy, I provide a set-cover-based algorithm that queries more efficiently to find a safe policy compared with other candidate methods (Ch. 7).

I contribute to formulating the well-studied avoiding negative side effects problem (Amodei et al. 2016) by modeling negative side effects as occupancy constraints. I use a querying approach to elicit more information from the human user to guarantee safety, while the literature mostly focuses on how the robot can find a safe or conservative policy to avoid side effects without communicating with the human user (Hadfield-Menell et al. 2017). In terms of the algorithms, I contribute to identifying a connection between the problem of querying to find a safe policy and set cover problems. So I modified the algorithms designed for set cover problems and use them for our problem.

Contribution III. When both reward uncertainty and safety-constraint uncertainty are present, it would be difficult to decide which type of query we want to pose, and what query to pose. One straightforward way is to re-use the methods in the previous contributions: We can find an optimal reward query and an optimal safety-constraint query and pose the one that is the more immediately useful. I provide an algorithm that uses batch-query-based heuristic to find empirically better queries than myopically choosing reward queries or safety-constraint queries (Ch. 8).

	no reward uncertainty	with reward uncertainty
no safety-constraint uncertainty	-	Contribution I
with safety-constraint uncertainty	Contribution II	Contribution III

Table 1.1: Contributions under different settings.

1.3 Approaches

Finding an optimal query is difficult. It can be equivalent to a combinatorial optimization problem, which is known to be NP-hard. The algorithms I provide exploit some properties of the query selection objectives. I briefly summarize the techniques used in this thesis.

- First, the objective function can be *submodular*. This is a nice property since we can use a greedy construction method to find a query with provable optimality bounds. I use this approach in Ch. 5 and Ch. 7.

- When the objective function is not submodular, a greedy construction method does not have an optimality guarantee. I will instead search the query space to find the optimal query, but also prune some known-to-be-suboptimal queries to improve the efficiency of the algorithm. I use this approach in Ch. 6.

- When the robot poses queries sequentially and only poses a single query at a time, it can pose queries based only on myopic information, which does not have good performance. Instead, the robot can find a *set* of useful queries and use them to guide its query selection. I use this approach in Ch. 8.

This thesis is structured as follows. I first describe background concepts in the background chapter (Ch. 2). I formally describe the problems in Ch. 3 and review the relevant literature in Ch. 4. I then report the results on querying under reward-only uncertainty (Ch. 5), querying under safety-constraint-only uncertainty (Ch. 6, 7), and querying under joint uncertainty (Ch. 8). Lastly, I conclude in Ch. 9 by summarizing the contributions and describing possible future work..

CHAPTER 2

Background

In this chapter, I describe some formal notations and fundamental concepts used in this thesis, including Markov decision processes (MDPs) and their variations. I also describe how an optimal policy is computed in these formulations. At the end, I briefly introduce the query semantics used in the following chapters.

2.1 Markov Decision Processes

In this section, I start by describing the concept of Markov decision process (Sutton and Barto 2018), which assumes the reward function is known and there are no safety constraints. Then I describe the variation under reward uncertainty, called *reward-uncertain MDPs*, and the variation when there are safety constraints present, called *constrained MDPs*.

Without reward uncertainty or any safety constraints, I model the domain that the robot deals with as a Markov decision process.

Definition 2.1. *A finite-state Markov decision process (MDP) (Sutton and Barto 2018) is a tuple $\langle S, A, T, r, \alpha, \gamma \rangle$, with finite state space S, finite action space 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; $\alpha \in \Delta_{|S|}$ is the initial state distribution and $\gamma \in [0, 1]$ is the discount factor.*

Note that I assume the state space and the action space are both finite (constrained by the techniques I will use in this thesis). The robot's decision-making is represented by a *policy*, denoted by $\pi : S \times A \to [0, 1]$; $\pi(s, a)$ is the probability that the robot takes action a in state s. The quality of a policy under a reward function r is represented by the *value* of the policy, denoted by $V_r^\pi = \mathbb{E}[\gamma^t r(s_t, a_t) | \alpha; s_t, a_t \sim \pi]$, which is the expected cumulative

rewards by following policy π under reward function r. An *optimal policy* under reward r is the policy that has the largest value: $\pi_r^* = \arg\max_\pi V_r^\pi$. I denote its value by V_r^*. When the reward function is unambiguous from the context, I omit the reward function in these notations (using V^π, π^*, V^* instead).

There are dynamic programming algorithms that find optimal policies for finite-state MDPs like value iteration and policy iteration (Watkins and Dayan 1992). In this thesis, I will need to add constraints to the robot's policy (described in the following sections). I instead use a linear programming formulation (Farias and Van Roy 2003) so that I can easily add constraints to it. The 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, discounted by γ. Formally, $x(s,a) = \mathbb{E}\big[\gamma^t \mathbf{1}[s = s_t, a = a_t] \big| \alpha; s_t, a_t \sim \pi\big]$, where $\mathbf{1}[\cdot]$ is the indicator function. The following linear programming problem finds the occupancy measure of the optimal policy:

$$\max_x \sum_{s,a} x(s,a) r(s,a) \tag{2.1}$$

$$\text{s.t.} \sum_{a'} x(s',a') = \gamma \sum_{s,a} x(s,a) T(s,a,s') + \alpha(s'), \forall s'. \tag{2.2}$$

Eq. 2.2 is a flow-conservation constraint that encodes the transition function. To recover a policy from its occupancy, we simply find $\pi(s,a) = \frac{x(s,a)}{\sum_{a' \in A} x(s,a')}$ when $\sum_{a' \in A} x(s,a') > 0$. The policy does not occupy states with $\sum_{a' \in A} x(s,a') = 0$, and so is not defined in these states.

2.1.1 Factored MDPs

A factored MDP (Boutilier, Dean, and Hanks 1999) is an MDP with an additional assumption on the state space. It assumes that a state can be represented by a set of features. For example, in the domestic robot example, a state can be represented by the location of the robot, the location of a drink, the states of vases (intact or broken), etc. Each feature can have a finite number of values. So the state space is the cross product of all possible values of all the features.

I will use Φ to denote a set of features and ϕ as one feature. Let $\phi_1, \phi_2, \ldots, \phi_n$ be a set of features, a state s can be represented by the values of these features: $s = \{\phi_1 = i_1, \phi_2 = i_2, \ldots, \phi_n = i_n\}$. For example, a state can be $\{robot\_location = location\_A, drink\_location = location\_B, vase\_A = intact\}$. I will formulate violating safety constraints as changing the values of certain features to be different from their

initial values (for example, $vase\_A$ is changed to $broken$ from $intact$). I will formally introduce how safety constraints are formulated as features in Ch. 3.

2.1.2 Reward-Uncertain MDPs

When we model the domain as an MDP, we assume the reward function is known by the robot. In reality, the robot may be uncertain about the true reward function that the human wants it to optimize. So we model the domain as a reward-uncertain MDP (Regan and Boutilier 2009).

Definition 2.2. *A reward-uncertain MDP is a tuple, $\langle S, A, T, \mathcal{R}, \alpha, \gamma \rangle$. The elements have the same interpretation as in the standard MDP definition, except instead of a reward function r, we have a set of* reward candidates*, $\mathcal{R} = \{r_1, \ldots, r_n\}$. There is a true reward function r^* that is only known by the human.*

As in the general preference elicitation setting, I assume that $\mathcal{R}$ contains the set of all possible reward functions that the human may have, which contains the true reward function r^*. After querying, the robot should be closer to finding the true reward function. This assumption is accepted in the literature (Ramachandran and Amir 2007; Viappiani and Boutilier 2010).

Assumption 2.1. *The true reward function r^* is in $\mathcal{R}$.*

Since I consider a Bayesian setting, I also assume that the robot has a prior distribution over the set of possible reward functions $\psi \in \Delta_{|\mathcal{R}|}$. I denote by $\mathbb{P}[r = r^*; \psi]$, or simply $\mathbb{P}[r; \psi]$, the probability that a reward function $r \in \mathcal{R}$ is the true reward function. It is known that, without querying the human to gain more information about r^*, the optimal policy under reward belief ψ is the same as the optimal policy under the *mean reward function* $\bar{r}_\psi$, where $\bar{r}_\psi(s, a) = \sum_{r \in \mathcal{R}} \mathbb{P}[r; \psi] r(s, a), \forall s \in S, a \in A$ (Theorem 3 in Ramachandran and Amir (2007)). The key to the proof of the claim is the following observation: If we consider the objective in Eq. 2.1, we have

$$\max_x \mathbb{E}_{r \in \mathcal{R}; \psi} \sum_{s,a} x(s, a) r(s, a) \tag{2.3}$$

$$= \max_x \sum_{s,a} x(s, a) \mathbb{E}_{r \in \mathcal{R}; \psi} r(s, a) \tag{2.4}$$

$$= \max_x \sum_{s,a} x(s, a) \bar{r}_\psi(s, a), \tag{2.5}$$

which is a result of linearity of expectation. So we can find the optimal policy under ψ using Eq. 2.1 by replacing r with $\bar{r}_\psi$, which becomes the following linear programming problem:

$$\max_x \sum_{s,a} x(s,a)\bar{r}_\psi(s,a) \tag{2.6}$$
$$\text{s.t.} \sum_{a'} x(s',a') = \gamma \sum_{s,a} x(s,a)T(s,a,s') + \alpha(s'), \forall s'.$$

This means that finding the optimal policy under a distribution of reward functions is as difficult (or easy) as finding the optimal policy under a single reward function. I will discuss, when querying is allowed, how to reduce such reward uncertainty to improve the performance the most in Ch. 5.

2.1.3 Constrained MDPs

In the previous section, I considered the setting where the robot is uncertain about the true reward function. In this section, I consider how the robot follows safety constraints and avoids visiting unsafe states.

For now, let us assume the robot knows a set of states, $S' \subseteq S$, that it should avoid. Semantically, S' is the set of states which may negatively surprise the human (a vase is broken, a carpet is made dirty, etc.). We can represent such constraints as the following linear constraints (which is the formulation used in Altman (1999)):

$$\sum_a x(s,a) = 0, \forall s \in S'. \tag{2.7}$$

By adding this constraint to Eq. 2.1, we have the following linear programming problem:

$$\max_x \sum_{s,a} x(s,a)r(s,a) \tag{2.8}$$
$$\text{s.t.} \sum_{a'} x(s',a') = \gamma \sum_{s,a} x(s,a)T(s,a,s') + \alpha(s'), \forall s' \tag{2.9}$$
$$\sum_a x(s,a) = 0, \forall s \in S'. \tag{2.10}$$

By imposing the constraint on occupancy (Eq. 2.10), the value of the optimal policy would be monotonically less than the value of the optimal policy without such constraints. If the robot is uncertain about whether it should avoid all states in S', it can query the human to

verify (Ch. 6 and 7). Also note that this linear programming problem may no longer have feasible solutions. For example, S' may contain states that the robot has to visit regardless of what policy it follows. I will also discuss how to find an initial feasible policy, if one exists, by querying the human (Ch. 7).

2.2 Query Selection Problem

In this thesis, the robot communicates with the human by posing queries and receives the human's responses. The technical contribution is efficiently finding the optimal query. Query selection problems are studied in the literature in both single-shot decision making and sequential decision making settings. In this section, I introduce the general query selection problem considered in the literature (Cohn 2016; Viappiani and Boutilier 2010).

First, I assume that both the human and the robot know the space of all possible queries and a *response model*. A response model means how a query should be answered under true preferences. I illustrate the idea using *k-policy queries*, defined below.

Definition 2.3. *A k-policy query contains a set of k policies, $q = \{\pi_1, \pi_2, \ldots, \pi_k\}$. The response model is that the human responds with the policy in this set that has the largest value under the true reward function r^*.*

This response model is represented by the notation $q \rightsquigarrow \pi \implies V_{r^*}^{\pi} \geq V_{r^*}^{\pi'}$ for $\pi' \in q$, where we use $q \rightsquigarrow \pi$ to denote the event that the human responds with π when query q is posed. One useful concept I adopt from the literature is the objective of query selection. Intuitively, the robot wants to find a query such that, after knowing the response to the query, the value of the posterior optimal policy is improved the most in expectation. This is called *expected posterior utility (EPU)* (Viappiani and Boutilier 2010):

$$EPU(q, \psi) = \sum_{d \in q} \mathbb{P}[q \rightsquigarrow \pi; \psi] V^*_{q \rightsquigarrow \pi; \psi}. \tag{2.11}$$

The goal is to find a query that optimizes the EPU function above under an initial reward belief ψ. Maximizing this objective is equivalent to maximizing the *expected value of information (EVOI)* (Viappiani and Boutilier 2010), defined below:

$$EVOI(q, \psi) = \sum_{d \in q} \mathbb{P}[q \rightsquigarrow d; \psi](V^*_{q \rightsquigarrow d; \psi} - V^*_{\psi}). \tag{2.12}$$

Note that the only difference between $EVOI$ and EPU is the subtracted term V^*_{ψ}, which is independent of q. Semantically, it measures how much the robot expects to improve the

value of its policy compared with not posing a query. We can see that $EVOI$ is always non-negative as a result of Jensen's inequality. We use EVOI if we need to compute the gain of posing a query. If there is a cost for posing a query, we can decide if it is worth posing the query by comparing its $EVOI$ value and the cost.

For some other problems, it makes more sense if the robot poses queries in a sequence and receives the human's response after it poses each query. Those problems do not pose k-response queries and I will define these problems later.

2.3 Combinatorial Optimization and Submodularity

Finding an approximately-optimal query in some settings relies on some useful results in the combinatorial optimization literature. In general, a combinatorial optimization problem defines an objective function $f : X \to \mathbb{R}$. We want to find a solution $S \subseteq X, |S| = k$ that maximizes the value of $f(S)$. We could exhaustively enumerate all k-subsets of X to find the optimal solution. However, this would evaluate $\binom{|X|}{k}$ subsets, which can be computationally expensive.

We could instead consider a greedy construction algorithm. Instead of evaluating all possible k-subsets, we greedily construct the solution by adding the element that increases the objective the most (Krause and Golovin 2014). This has the time complexity of $O(|X|k)$. Unfortunately, we cannot provide a guarantee on the quality of the solution pro-

Algorithm 2.1 Greedy Construction Algorithm (Krause and Golovin 2014)

1: $S_0 \leftarrow \emptyset$
2: **for** $i = 1, 2, \ldots, k$ **do**
3: $\quad x_i \leftarrow \arg\max_x f(S_{i-1} \cup \{x\})$
4: $\quad S_i = S_{i-1} \cup \{x_i\}$
5: **end for**
6: return S_k

vided by the greedy algorithm for arbitrary f. Prior work finds that if f satisfies a property called *submodularity* (Krause and Golovin 2014), then we can provide a guarantee on the solution found above.

Definition 2.4. *A function $f : X \to \mathbb{R}$ is* **submodular** *if for any $S \subseteq S' \subseteq X$ and any $x \in X$,*

$$f(S \cup \{x\}) - f(S) \geq f(S' \cup \{x\}) - f(S'). \tag{2.13}$$

Intuitively, for any element $x \in X$, a submodular function f has a property that adding x can help to marginally increase the value of f more if it is added *earlier* than *later*. Con-

cretely, if S is a subset of S', and both S and S' are subsets of X, the marginal increment in value of f is larger when adding x to S than adding x to S'.

Denote the solution found by the greedy algorithm by S^{greedy}. If the objective function f is submodular, we have the following guarantee.

Theorem 2.1. *(Krause and Golovin 2014) When f is submodular, the greedy algorithm guarantees to find a solution, S^{greedy}, such that $f(S^{greedy}) \geq (1 - \frac{1}{e})f(S^*)$, where e is the base of the natural logarithm and S^* is the optimal solution.*

This provides an approach to finding efficient algorithms that find queries with provable optimality guarantees. I exploit the submodularity property of the query selection objectives and design greedy algorithms in Ch. 5 and 7.

CHAPTER 3

Problem Statement

I have briefly described the problems considered in this thesis in the introduction (Ch. 1). In this chapter, I formally describe the problem formulations using the terminology defined in the background chapter (Ch. 2).

As described in Ch. 1, the robot may have uncertainty over the true reward function or what safety constraints it needs to satisfy. Before it executes a policy, the robot has a chance to query the human user and will receive responses to the queries. It expects to find a better policy after knowing the responses.

Concretely, there are three sequential phases in the interaction process:

- Reward design phase: The human specifies her preferences and may provide some initial information about what safety constraints need to be satisfied.
- Querying phase: The robot has a chance to query the human about the true reward function or the safety constraints and receives the human's response to the query (or queries).
- Execution phase: The robot executes a (safely-)optimal policy after querying. No more communication is possible in this phase.

For different problems, in the querying phase, the robot may either pose one query that contains k elements, or keep posing queries sequentially until a goal is met. The elements contained in a query and how the human responds to a query varies in different problems, which I will elaborate in the later chapters.

The robot wants to improve its performance under possible reward uncertainty and safety-constraint uncertainty. Importantly, when safety constraints are present, the robot needs to guarantee safety by not violating the human's unexpressed safety constraints. So querying may serve the following purposes: 1) It improves the value of the robot's policy by

reducing reward uncertainty; 2) it improves the value of the robot's policy by identifying which safety constraints the robot does not need to satisfy; 3) and it helps to find a safe policy when the robot initially does not know such a policy.

We could exhaustively search for the optimal query in the query space. The technical contributions of this thesis focus on how to efficiently find an (approximately-)optimal query.

I summarize the problems considered in this thesis as follows, as illustrated in Table 1.1.

1. Querying and planning under (only) reward uncertainty (Contribution I): Ch. 5 (Zhang, Durfee, and Singh 2017).

2. Querying and planning under (only) safety uncertainty (Contribution II): Ch. 6, Ch. 7 (Zhang, Durfee, and Singh 2018, 2020b).

3. Querying and planning under reward and safety uncertainty (Contribution III): Ch. 8.

The problem definitions of these problems are detailed in the following sections.

3.1 Efficiently Finding Approximately-Optimal Queries Under (Only) Reward Uncertainty

In this section, I focus on reward uncertainty without considering safety constraints, that is, I assume that there is no safety constraint that the human user cares about. The robot formulates the domain as a reward-uncertain MDP (defined in Sec. 2.1.2). It has a Bayesian prior over the reward candidates: It initially knows a set of reward candidates, $\mathcal{R}$, and a prior belief, ψ. The true reward function $r^* \in \mathcal{R}$ is only known by the human. The robot can pose *k-response queries*, which is like a multiple-choice question and the human responds with one choice.

I will first show that it is sufficient to only consider k-policy queries (Definition 2.3). This result is a generalization of Cohn (2016), which provides a similar result in a one-shot decision making setting. To repeat the definition here, a k-policy query contains a set of k policies, $q = \{\pi_1, \pi_2, \ldots, \pi_k\}$. The response model is that the human responds with the policy in this set that has the largest value under the true reward function r^*.

The goal is to select a k-policy query q that maximizes the expected posterior utility (EPU) (defined in Eq. 2.11),

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

Or equivalently, optimizing the expected value of information (Eq. 2.12),

$$EVOI(q, \psi) = \sum_{d \in q} \mathbb{P}[q \rightsquigarrow d; \psi](V^*_{q \rightsquigarrow d; \psi} - V^*_{\psi}).$$

I provide an algorithm that finds a provably approximately-optimal policy query by solving a sequence of mixed-integer linear programming problems. Additionally, since a human user may have difficulty understanding policy queries, I also consider finding trajectory queries using the information in the approximately-optimal policy query. I will discuss this problem in Ch. 5.

Non-Bayesian setting. My work assumes that the robot has a prior over the possible reward functions. We could remove this assumption and consider a minimax-regret objective. The setting is well explored in the literature (Regan and Boutilier 2009, 2011a) and I do not consider this setting. It is worth pointing out that they consider a minimax-regret criterion and find queries by first finding dominating policies, which informs my stragety for finding minimax-regret safety-constraint queries under a non-Bayesian setting (Ch. 6).

3.2 Efficiently Finding Optimal or Empirically Good Queries Under (Only) Safety-Constraint Uncertainty

In this section, I focus on safety-constraint uncertainty without reward uncertainty. The robot initially knows the set of all possible safety constraints that the human may have (for example, never breaking a vase, never making a carpet dirty, never moving a box, etc.). However, it only needs to follow a subset of them, depending on the human's preferences (for example, the human may not mind if the robot moves a box). Without querying, the robot should never violate any possible safety constraints so as to guarantee safety. The robot can only ignore and violate a possible safety constraint when it queries the human and confirms that the human does not require that safety constraint to be satisfied. The question is what safety constraints the robot should query about to find a safe policy or to improve a known safe policy.

I assume that the robot knows a reward function that contains incomplete or inaccurate reward information (the same assumptions are made in (Hadfield-Menell et al. 2017; Mindermann et al. 2018)). For example, in Example 1.2, the human may only say that there is a positive reward if the robot delivers a drink to the human, while saying nothing about other objects in the environment like carpets, boxes, etc. If the robot optimizes such a reward

function, it may have unsafe behaviors such as making a carpet dirty or moving away some boxes.

This motivates the definition of safety in this thesis, that is, the robot should not cause a side effect to the environment unless it is certain that it is permitted to do so. Concretely, I assume that the MDP is a factored MDP (defined in Sec. 2.1.1), and that the robot can partition the features into the following sets:

- $\Phi_F^{\mathbf{R}}$: The *free-features* (i.e., freely changeable). The robot knows that these features can be changed freely (for example, its location).
- $\Phi_L^{\mathbf{R}}$: The *locked-features*. The robot knows it should never change any of these features (for example, breaking a vase).
- $\Phi_?^{\mathbf{R}}$: The *unknown-features*. These are features that the robot does not (initially) know whether the human considers freely changeable or locked.

The human similarly partitions features, but only into the sets $\Phi_L^{\mathbf{H}}$ and $\Phi_F^{\mathbf{H}}$. I assume that the robot's knowledge, while generally incomplete ($\Phi_?^{\mathbf{R}} \neq \emptyset$), is consistent with that of the human. That is, $\Phi_F^{\mathbf{R}} \subseteq \Phi_F^{\mathbf{H}}$ and $\Phi_L^{\mathbf{R}} \subseteq \Phi_L^{\mathbf{H}}$. The robot's policy is **safe** if it does change any features that are known to be locked or are unknown.

Finding the *safely*-optimal policy can be formulated as the following problem.

$$\max_x \sum_{s,a} x(s,a) r(s,a) \tag{3.1}$$

$$\text{s.t.} \sum_{a'} x(s',a') = \gamma \sum_{s,a} x(s,a) T(s,a,s') + \alpha(s'), \forall s' \in S \tag{3.1a}$$

$$\sum_{a \in A} x(s,a) = 0, \forall s \in S_{\Phi_L^{\mathbf{R}} \cup \Phi_?^{\mathbf{R}}} \tag{3.1b}$$

where S_Φ is the set of states where one or more features in Φ have different values from the initial state. I formulate safety-constraint uncertainty as uncertainty over which category an unknown feature is in ($\Phi_L^{\mathbf{H}}$ or $\Phi_F^{\mathbf{H}}$). So the robot can query about if a feature ϕ is in $\Phi_L^{\mathbf{H}}$ or $\Phi_F^{\mathbf{H}}$. Whenever the robot is told that an unknown feature is free, it is allowed to change that feature (by removing the constraint on that feature in Eq. 3.1b) so that the objective value monotonically increases.

I consider the query selection problems in two scenarios: the robot initially finds a safe policy or the robot initially cannot find a safe policy.

3.2.1 Improving a Safe Policy

For this query selection problem, I make an additional assumption that the robot knows a safe policy under the initial partition, $\Phi_L^{\mathbf{R}}, \Phi_F^{\mathbf{R}}$, and $\Phi_?^{\mathbf{R}}$. In the querying phase, the robot can ask about k unknown features and the human responds with which features are free/locked. I provide an algorithm that finds a *minimax-regret* query to improve the robot's policy the most. I define the maximum regret of a query to be how a different query can outperform this query in the worst case. The minimax-regret query minimizes the maximum regret. To avoid exhaustively evaluating all possible queries, I exploit some structures in the uncertainty space to prune some queries that are known-to-be suboptimal. I will define the objective in detail and describe my algorithm in Ch. 6.

3.2.2 Finding an Initial Safe Policy

The section above assumes that there exists a known safe policy and the purpose of querying is to find a safe policy with a higher value. I also consider the case where the robot initially does not know a safe policy. But it knows the probability that the human believes an unknown feature is free. The robot can sequentially query about the category of any unknown feature until it 1) finds a safe policy, or 2) proves that no safe policy exists. The objective is to reach either outcome using the minimum number of queries in expectation.

In this problem, I find that finding a safe policy and proving that no safe policy exists each corresponds to a set cover problem. I contribute to designing a query selection algorithm that exploits this structure by simultaneously solving two set-cover problems. I will discuss this problem in Ch. 7.

3.3 Low-Cost Querying Under Both Reward and Safety-Constraint Uncertainty

So far, I considered the cases where the robot has only reward uncertainty or only safety-constraint uncertainty. Now I consider a more general case where the robot has both types of uncertainty. The robot has reward uncertainty: It initially knows a prior belief over a set of reward candidates, $\mathcal{R}$, which contains the true reward function (the same as Sec. 3.1). Moreover, a state is represented by a set of features, partitioned into locked, free, and unknown features. The robot is not allowed to change any locked features and is uncertain about if unknown features are changeable or not (the same as Sec. 3.2).

After querying, the robot is evaluated by its policy under the true reward function while it guarantees that it does not violate any unexpressed safety constraints.

In this query selection problem, the robot is allowed to pose both reward queries and feature queries. I assume that there is a cost to pose a query. So the robot needs to compromise between the value of the safe policy after querying and the cost of querying. The robot sequentially poses either a reward query or a feature query, and it decides when to stop. We can clearly use the algorithms designed for the problems in the previous sections. A simple heuristic is to myopically find the best a reward query and the best feature query, and pose the one with the higher EVOI. However, we will see in some simple examples that such a heuristic has clear disadvantages. I provide a method where the robot imagines what query it would pose if it is allowed to pose a batch query. It uses such a hypothetical batch query to guide its query selection. I find such a heuristic has better performance on average compared with some baseline heuristics. I will discuss this problem in Ch. 8.

3.4 Discussion on the Querying Semantics

To avoid overly consuming the human user's cognitive load, I make the assumption that the robot can pose a query that contains no more than k elements as a multiple-choice question, or minimizes the number of queries needed to achieve an objective. This assumes that each query or each element in the query has the same cognitive cost. However, if we have more information about the cognitive cost of the human to answer a query, we could generalize the constraint on the elements in the query or the number of queries to be a cognitive budget. The robot can still use the query selection algorithms like Algorithm 5.1 (described in Ch. 5) in this dissertation, but the constraint becomes that the sum of costs of querying does not exceed the cognitive budget.

There are other settings we can consider. For example,

(a) querying before the robot executes a policy versus querying during the execution of the policy;

(b) when the robot queries before its execution of a policy, it can either pose all the queries in one batch or in a sequence (that is, the robot receives the human's response to its posed queries before it poses the next query).

Regarding (a), I assume that the robot only poses queries before it takes any actions in the environment. The human would attend the robot's query or queries only once, regardless if the robot poses queries in a batch or in a sequence. When the robot queries in a sequence, I assume that the robot poses the next query (almost) immediately after it receives the

human's response to the previous query, so that there is no context switch for the human. So querying before the robot executes its policy distracts the human user only once, and the human would know when she expects questions from the robot. Research has shown that an interruption may cause stress and frustration to the human and make her less productive (Mark, Gudith, and Klocke 2008; Mark et al. 2016). So human users generally prefer fewer interruptions.

A different setting would be that the robot is allowed to pose one query at any time during its execution. In this case, to answer the question of when is the best time to pose a query, we can evaluate the gain of posing a query now versus later (like expected myopic gain in Cohn et al. (2010)). We can use an outer loop to decide when to pose a query. If it is worth querying in the current state, we then use the query selection algorithm in this dissertation to find a good query to pose.

Regarding (b), when I constrain the number of elements in a query (for example, the k-policy query in Definition 2.3), I assume that the robot would pose the query in one batch. However, when the robot can ask about k policies, it may be more helpful to pose queries in small batches and in a sequence. For example, it may pose m-policy queries for $\lfloor k/m \rfloor$ times. Cohn (2016) has discussed that it is computationally expensive to find the optimal query policy and we need to consider *policy set queries* to make sure that the query policy space contains the optimal query policy. This is beyond the scope of this dissertation. We can still pose approximately-optimal m-policy queries found using the method in this dissertation, while such a query selection method would not take possible future queries into consideration.

CHAPTER 4

Related Work

In this thesis, the robot obtains information about the human's preferences through querying. There are also other ways of communication studied in the literature. In this chapter, I will describe the related work in the literature of human-agent interaction, preference elicitation, and reward design. In the querying under safety-constraint uncertainty setting, the problem of guaranteeing safety is relevant to the AI safety literature, which I will also review in this chapter.

4.1 Human-Agent Interaction and Preference Elicitation

Depending on the level of the robot's automation or the degree of the human's involvement, we can categorize the related work into the following.

1. Teleoperation: The human fully controls the robot. The robot is not autonomous at all and does not necessarily know the human's preferences.

2. Supervised operation: The human supervises the robot and provides feedback when necessary (Hadfield-Menell et al. 2016a; Torrey and Taylor 2013). The human does not need to control the robot, but only needs to provide feedback.

3. "Consultative" operation: The human does not actively supervise the robot. The robot is expected to actively query the human about its uncertainty. The human does not need to attend the robot during its execution. She is only available to answer the robot's queries (Akrour, Schoenauer, and Sebag 2012; Viappiani and Boutilier 2010). *This thesis is in this category.*

4. Autonomous operation: The human does not supervise the robot and there is no communication between the human and the robot. The robot operates fully autonomously.

The problems I consider are in the third category: The robot is close to fully-autonomous except it poses limited queries to improve its performance or to guarantee safety. I also assume that the queries can be communicated directly rather than using actions in the environment. When such a protocol is absent, the robot may be able to use environmental actions to elicit reward information by assuming that the human can be part of the environment. For example, in Sadigh et al. (2016), an autonomous driving car can nudge to an adjacent lane to observe other human drivers' reactions and decide what types of drivers they are.

Note that my work is not the only one in the third category. Viappiani and Boutilier (2010) consider the problem of optimal query selection under reward uncertainty, but for single-shot decision-making problems. Their contribution is providing a greedy algorithm which runs linearly in the number of decisions. These methods cannot be applied to our problem directly since we cannot afford to evaluate all policies (corresponding to decisions in our setting). Cohn (2016) built upon the previous work and provided a useful result: It is sufficient to only consider querying about decisions to find an optimal multiple-response query. This result inspires my work in Ch. 5.

Another dimension is how the robot obtains the information about the human's preferences. The robot may initially have a prior belief over the human's preferences. There are at least three possible ways to gain more preference information as listed below.

1. The human demonstrates how she performs the task by providing trajectories of a nearly-optimal policy (Abbeel and Ng 2004; Hadfield-Menell et al. 2016b; Ramachandran and Amir 2007).

2. The human provides feedback to the robot when she supervises the robot.

3. The human answers the robot's queries when the robot poses them.

I consider bullet 3 in this thesis. I do not assume the human would provide demonstrations (bullet 1) or will actively give feedback (bullet 2), which may require much more effort than answering the robot's questions. Note that these sources of preference information are not necessarily disjoint. The robot may gain information from multiple sources (for example, both bullet 1 and 3 in (Palan et al. 2019)).

Some other related works focus on finding high-quality queries in large or continuous domains, without providing theoretical guarantees on the quality of the queries (Akrour, Schoenauer, and Sebag 2012, 2013). For example, the APRIL algorithm (Akrour, Schoenauer, and Sebag 2012) considers trajectory queries, but uses a sequential querying process where, instead of presenting k trajectories at once, it presents one trajectory at a time and

essentially asks the human a binary ($k = 2$) query. The nature of a query response might differ, such as when the human can answer by manipulating the robot to demonstrate a desired trajectory, rather than selecting from among offered trajectories (Jain et al. 2013). Mindermann et al. (2018) consider reward queries, where they ask the user which reward function in a small set of reward functions is closest to what she wants to specify. They use a greedy construction method to avoid an exponential number of subsets, although, unlike my work, they do not claim any optimality guarantee on the query found by their greedy algorithm. Christiano et al. (2017) consider sequential trajectory-comparison queries. They find queries by sampling a large set of trajectories and find the subset of them that can serve as the best query. I will consider generating queries using sampling as well (in Ch. 5), which is outperformed by an efficient greedy construction algorithm in my setting.

4.2 Reward Design

Amodei et al. (2016) made the observation that the accuracy of reward signals and the costs of obtaining such signals need to be balanced. For example, a reward function that is a function of observations may be easy to define but inaccurate, like using the drinks the robot brings as a reward signal. Such a reward function may not capture possible side effects the robot may cause. On the other hand, asking the human to always evaluate the robot's policy would be accurate but costly. So the robot's objective is to query to obtain reward signals to improve its performance while not increasing the human's burden too much. The difficulty of designing the correct reward function motivates works on *inverse reward design*. In Hadfield-Menell et al. (2017) and Milli et al. (2017), the robot regards the designed reward function as an *observation* and aims to infer the true reward function that the human failed to specify. In Hadfield-Menell et al. (2017), the human defines a reward function *correctly* in an MDP in her mind, called the *training MDP*. She makes sure that the optimal policy under such a reward function is desirable in the training MDP. However, the environment where the robot's performance is evaluated, called the *testing MDP*, may be different. If the robot simply optimizes the human-designed reward function, it may cause side effects to the environment or gain rewards in an unexpected way. Different from my work, their work focuses on the robot's reasoning about the true reward function rather than communicating with the human.

Other works consider if a reward function is the best form of specifications for preferences. It may be challenging for a human to figure out the correct rewards for all the states so that the optimal policy is desirable. It may be easier to specify a logic formula that the robot has to follow. Alshiekh et al. (2018) represented the human user's preferences as

linear temporal logic formulas. If the human's safety preference can be any logic formula, the space of possible logic formulas can be infinite and it may be impractical to find the human's exact preference using querying. So I consider simplified logic formulas which are not as expressive as in Alshiekh et al. (2018) in this thesis.

4.3 AI Safety

The safety constraints I consider in this thesis are related to the AI safety literature. Most work in the literature considers developing safety-aware agents without communication between the robot and the human. Here I only provide a brief survey on safety in MDPs. I refer interested readers to Amodei et al. (2016), Garcıa and Fernández (2015), and Leike et al. (2018, 2017) for more thorough surveys on AI safety.

Safety as rewards. A common approach to make the robot's behavior safe is to design or infer a *safety reward function*. Amin, Jiang, and Singh (2017) assumed that the true reward function is a sum of a task-independent safety reward function and a task-specific reward function. The robot can observe the human's optimal policies in multiple MDPs and infer the safety reward function.

The robot can also add a side-effect penalty to its reward function (Krakovna et al. 2018). Concretely, after taking an action, the robot can compute the action's impact to the environment compared to a baseline (for example, many states can no longer be reachable after taking such an action). They proposed several possible baselines and several criteria to measure an action's impact, and did empirical evaluations on how much they help to reduce side effects. Turner, Hadfield-Menell, and Tadepalli (2020) approached this problem using multiple reward functions. They assumed that the robot initially knows a set of reward functions that it may need to optimize in the future. When it optimizes the current reward function, the robot wants to make sure that it would not impair the values of the optimal policies under different reward functions in the future. For example, the robot is rewarded to turn off a switch, but it may also be asked to hand over a vase to the human in the future. If the robot breaks the vase on the way to turn off a switch, it would be impossible to find a policy that has a high return of handing over a vase (which is already broken).

Shah et al. (2019) observed that the initial state contains the information of what the human tries to maintain. For example, if a vase is not broken in the initial state but it is very easy to break the vase by taking any policy, then it means that the human is trying not to break the vase. So the robot can infer a safety reward function from the initial state. I

also exploit the information in the initial state in my work by keeping the unknown features unchanged unless permitted by the human.

These works show that their algorithms successfully avoid causing side effects in some tabular-world domains (importantly, without encoding any domain-specific information about side effects). However, there is no guarantee in general on if the robot's behavior is safe in untested domains.

Safety as constraints. Achiam et al. (2017) and Chow et al. (2017) considered the learning problem in the setting where the robot is required to satisfy the constraints in the form of $\mathbb{E}_{s,a\sim\pi}c_i(s,a) \leq d_i$, where c_i is a cost function $c_i : S \times A \to \mathbb{R}$ and d_i is a known constant. Other similar works focus on planning to find policies that satisfy some constraints/commitments or maximize the probability of reaching a goal state (Kolobov, Mausam, and Weld 2012; Teichteil-Königsbuch 2012; Witwicki and Durfee 2010; Zhang, Durfee, and Singh 2020a).

This thesis formulates a "safety requirement" as a kind of constraint, so it falls into this category. The robot guarantees that it does not surprise the human by guaranteeing that it will not violate any safety constraints. Also, the related work mostly assumes that safety constraints are known *a priori*. I consider the case where the robot is uncertain about which of the possible constraints are violable. The robot queries to figure out which possible constraints do not need to be satisfied.

CHAPTER 5

Efficiently Finding Approximately-Optimal Queries Under (Only) Reward Uncertainty

In this chapter, I consider the problem of querying under (only) reward uncertainty as specified in the problem statement (Sec. 3.1). Recall that the robot may be uncertain about the true reward function in the human's mind (r^*), while the robot has access to a set of possible reward functions ($\mathcal{R}$) and an initial belief over these reward functions (ψ). I assume that $r^* \in \mathcal{R}$.

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

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

new method to greedily construct an approximately optimal policy query without enumerating all policies.

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

5.1 Expected Posterior Utility and Expected Utility of Selection

I briefly described the query selection problem and k-policy queries in Section 2.2. In this section, I formally define k-response queries generally and the objectives of query selection.

I first consider a non-sequential decision-making problem: The robot has a space of decisions, $D = \{d_1, d_2, \ldots, d_m\}$, that it is choosing between; in our setting, each decision is a candidate plan (policy) to pursue. The utility of a decision $d \in D$ under a reward candidate $r \in \mathcal{R}$ is denoted by V_r^d. The robot wants to pick the decision with the highest utility given r^*, but since it only has probabilistic knowledge ψ about which reward candidate is r^*, it maximizes the expected utility given ψ. Communication between the human and robot has the robot offering a choice over some finite number k decisions, and the human responding with the decision with the largest utility value under her true reward function. Usually, $|D|$ is too large to offer all possible choices, so the challenge is to select a (often much) smaller subset of k decisions to best elicit the human's preferences.

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

The goal of the robot is to find a q that maximizes its **Expected Posterior Utility**

(EPU), defined in the background chapter (Eq. 2.11).

$$EPU(q, \psi) = \sum_{d \in q} \mathbb{P}[q \rightsquigarrow d; \psi] V^*_{q \rightsquigarrow d; \psi} \tag{5.1}$$

where $V^*_{q \rightsquigarrow d; \psi} = \max_{d' \in D} V^{d'}_{q \rightsquigarrow d; \psi}$, and $q \rightsquigarrow d; \psi$ is the posterior belief about the reward function after observing $q \rightsquigarrow d$, with the prior belief ψ. I define how to compute it later. A (k-response) decision query that optimizes the function above is called the *optimal (k-response) decision query*. The optimal decision query maximizes the utility of the optimal decision after knowing the query response in expectation.

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

$$EUS(q, \psi) = \sum_{d \in q} \mathbb{P}[q \rightsquigarrow d; \psi] V^d_{q \rightsquigarrow d; \psi} \tag{5.2}$$

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

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

$$\mathbb{P}[r | q \rightsquigarrow j; \psi] \sim \mathbb{P}[q \rightsquigarrow j | r] \mathbb{P}[r; \psi] \tag{5.3}$$

5.2 Finding Policy Queries

The preceding section provides a straightforward but expensive strategy for finding an optimal query: For every query that could conceivably be asked, compute its EPU (Eq. 2.11), which for each query involves computing the probability of each response, the posterior beliefs due to the response, and the optimal policy to follow given those beliefs. Then ask the query with the highest EPU. I now consider ways to find (approximately) optimal queries with much less computation.

A fundamental result that I use is that, in a general preference-elicitation setting, a robot can, without loss of optimality, restrict the choices offered in k-response queries to only the *decisions* it can actually act on (Cohn, Singh, and Durfee 2014). Since the robot is deciding among policies, this means that, in reward-uncertain MDPs, a robot can focus only on k-response *policy* queries (defined in Eq. 2.3). Formally, a policy query, $q = \{\pi_1, \ldots, \pi_k\}$, is a set of policies, and the response is such that $q \rightsquigarrow \pi \implies V^{\pi}_{r^*} \geq V^{\pi'}_{r^*}, \forall \pi' \in q$.

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

Theorem 5.1. *(adapted from (Cohn, Singh, and Durfee 2014)) For a reward-uncertain MDP, the optimal k-response policy query is the optimal k-response query.*

$$\max_{q \in \Pi^k} EPU(q, \psi) = \max_{q \in Q_k} EPU(q, \psi) \tag{5.4}$$

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

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

Although the robot can, without loss of optimality, limit its query search to k-response policy queries, there are still many such queries because the space of k-ary policy query candidates is of size $\binom{|\Pi|}{k}$. If we only consider deterministic policies, this number is $\binom{|A|^{|S|}}{k}$. So it is computationally expensive to evaluate all k-policy queries even in small domains. I need a more efficient way to find an approximately optimal policy query.

5.2.1 Greedy Construction of Policy Queries

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

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

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

In a general preference elicitation setting, Viappiani and Boutilier (2010) proposed a greedy construction method for finding an approximately optimal query, similar to the general greedy construction algorithm described in Algorithm 2.1. Instead of finding the k elements in the query altogether, the algorithm incrementally adds decisions to the query. Concretely, in the first iteration, $q_1 = \{d^*_\psi\}$ contains the optimal decision under prior belief ψ. In later iteration i, where $i = 2 \ldots, k$, the algorithm finds $q_i = q_{i-1} \cup \{d_i\}$ where $d_i = \arg\max_{d \in D} EUS(q_{i-1} \cup \{d\}, \psi)$. This algorithm performs k passes over the decision space, so its computational complexity is $O(|D|k)$. In our setting, this is $O(|\Pi|k)$, which is impractical given the size of Π.

Algorithm 5.1 Greedy Construction of Policy Queries

1: $q_1 = \{\pi^*_\psi\}$
2: **for** $i = 2, 3, \ldots, k$ **do**
3: $\quad q_i = q_{i-1} \cup \{\pi_i\}$
4: $\quad \pi_i = \arg\max_\pi EUS(q_{i-1} \cup \{\pi\}, \psi)$
5: **end for**
6: return q_k

I now define our variation of this process for reward-uncertain MDPs that avoids examining all of Π. Initially, $q_1 = \{\pi^*_\psi\}$, containing the optimal policy under ψ. In iteration i, $q_i = q_{i-1} \cup \{\pi_i\}$ where $\pi_i = \arg\max_\pi EUS(q_{i-1} \cup \{\pi\}, \psi)$. The psudocode of this algorithm is in Algorithm 5.1. Consider the difference between $EUS(q_{i-1} \cup \{\pi\}, \psi)$ and

$EUS(q_{i-1}, \psi)$.

$$\begin{aligned}
&\max_{\pi} EUS(q_{i-1} \cup \{\pi\}, \psi) - EUS(q_{i-1}, \psi) \\
=&\max_{\pi} \sum_{r \in \mathcal{R}} \mathbb{P}[r; \psi](\max_{\pi' \in q_{i-1} \cup \{\pi\}} V_r^{\pi'} - \max_{\pi' \in q_{i-1}} V_r^{\pi'}) \\
=&\max_{\pi} \sum_{r \in \mathcal{R}} \mathbb{P}[r; \psi] \max(V_r^{\pi} - \max_{\pi' \in q_{i-1}} V_r^{\pi'}, 0) \qquad (5.5)
\end{aligned}$$

Eq. 5.5 is what I want to optimize at each iteration to find π_i. One step of the procedure is illustrated in Figure 5.1. In the i-th iteration, we have $q_{i-1} = \{\pi_1, \ldots, \pi_{i-1}\}$. I find a policy π_i that maximizes the dark shaded area in the right-most figure, which is the weighted (by ψ) margin that the added policy outperforms the policies already in the query.

The maximum operator inside the objective function makes this equivalent to a mixed integer linear programming (MILP) problem. Based on Eq. 2.6, I use x to denote the state-action occupancy measure of π_i, and rewrite the optimization problem in Eq. 5.5 as follows.

$$\max_{x, \{y_r\}, \{z_r\}} \sum_{r \in \mathcal{R}} \mathbb{P}[r; \psi] y_r \qquad (5.6)$$

$$\text{s.t.} y_r \leq \sum_{s,a} [x(s,a) r(s,a)] - \max_{\pi \in q_{t-1}} V_r^{\pi} + M(1 - z_r), \forall r \in \mathcal{R} \qquad (5.6a)$$

$$y_r \leq 0 + M z_r, \forall r \in \mathcal{R} \qquad (5.6b)$$

$$\sum_{a'} x(s', a') = \sum_{s,a} x(s,a) T(s,a,s') + \alpha(s'), \forall s' \qquad (5.6c)$$

$$z_r \in \{0, 1\}, \forall r \in \mathcal{R} \qquad (5.6d)$$

I rewrite the $\max$ function of Eq. 5.5 as two constraints (corresponding to lines 5.6a and 5.6b), where at any given time only one of them applies. M is an arbitrarily large positive number. I introduce binary variables z_r for each $r \in \mathcal{R}$ (defined in line 5.6d) so that when $z_r = 0$, the first constraint (line 5.6a) is relaxed, since $y_r \leq M$ holds anyway, and the second constraint (line 5.6b) is enforced. When $z_r = 1$, the first constraint is enforced and the second constraint is relaxed. This is to enforce the constraint that $y_r \leq \max\{\sum_{s,a}[x(s,a)r(s,a)] - \max_{\pi \in q_{t-1}} V_r^{\pi}, 0\}$. The constraints in line 5.6c ensure that x is a valid state-action occupancy.

Analysis. The EUS of an approximately-optimal policy query from this greedy process enjoys the guaranteed performance $EUS(q_k, \psi) \geq \eta EUS(q^*, \psi) = \eta EPU(q^*, \psi)$, where

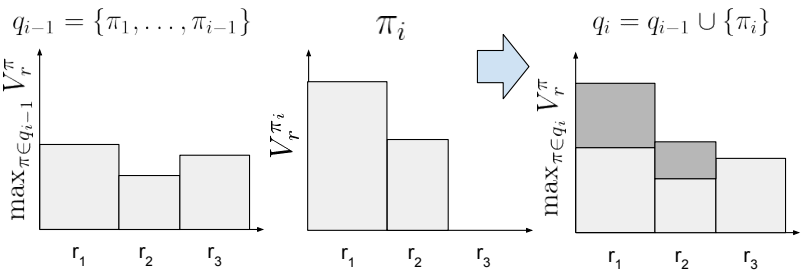


Figure 5.1: Greedy policy construction illustration.

$\eta = 1 - (\frac{k-1}{k})^k$ and q^* is the optimal k-response policy query (Viappiani and Boutilier 2010). Building a policy query with k elements requires solving k MILP problems. The complexity is polynomial in the number of state and action pairs, but combinatorial in the number of reward candidates in the worst case. For problems with many reward candidates, trying other approximation methods like policy gradient methods is left for future work.

Now I consider how to characterize the policies I found by solving the optimization problem in Eq. 5.6. At the i-th iteration, let $x, \{y_r\}, \{z_r\}$ be solutions. π_i is the policy that occupancy x represents. Then:

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

That is, π_i maximizes the mean reward $\sum_{r \in \mathcal{R}} \mathbb{P}[r; \psi] z_r r$, which is the mean reward of a subset of the reward candidates. Note that this is not a way to find π_i, since $\{z_r\}$ are outputs of the optimization problem. The MILP problem decides the mean reward of which subset of reward candidates to optimize. We may hypothesize that the policies in an *optimal* recommendation set are also optimal policies under some mean reward functions. This is verified by the following theorem.

Theorem 5.2. *Any policy in an optimal recommendation set must be the optimal policy under the mean reward of a subset of reward candidates.*

Proof. Suppose q^* is an optimal recommendation set, $EUS(q^*) = EPU(q^*) \geq EPU(q'), \forall q'$. To prove the theorem by contradiction, suppose $\pi \in q^*$ is not an optimal policy under the mean reward of any subset of reward candidates.

Let $\pi \in q^*$ dominate the reward candidates $R' \subseteq \mathcal{R}$, which means $V_r^{\pi} \geq V_r^{\pi'}, \forall r \in R', \pi' \in q^*$. Let $\hat{\pi} = \arg\max_{\pi'} V_{\bar{R}'}^{\pi'}$, where $\bar{R}' = \frac{\sum_{i:r_i \in R'} \psi_i r_i}{\sum_{i:r_i \in R'} \psi_i}$.

I consider the query $q' = q^* \setminus \{\pi\} \cup \{\hat{\pi}\}$.

$$\begin{aligned} EUS(q') &= \sum_{r \in \mathcal{R}} \mathbb{P}[r;\psi] \max_{\pi' \in q'} V_r^{\pi'} \\ &= \sum_{r \in R'} \mathbb{P}[r;\psi] V_r^{\hat{\pi}} + \sum_{r \notin R'} \mathbb{P}[r;\psi] \max_{\pi' \in q^*} V_r^{\pi'} \\ &> \sum_{r \in R'} \mathbb{P}[r;\psi] V_r^{\pi} + \sum_{r \notin R'} \mathbb{P}[r;\psi] \max_{\pi' \in q^*} V_r^{\pi'} \\ &= EUS(q^*) \end{aligned}$$

So q^* is not the optimal recommendation query. We have a contradiction. □

Thus, to construct the approximately optimal k-response policy query, iteratively apply the MILP (Eq. 5.6) k times to generate k subsets of the reward candidates, and then find the optimal policy for the mean reward function of each subset. The query is composed of these k policies, and, importantly, the process avoids searching over all policies.

5.3 Finding Trajectory Queries by Projection

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

$$q = \arg\min_{q \in Q} H(q_{\Pi}^* | q) \tag{5.7}$$

where $H(q'|q)$ is the conditional entropy, defined as $H(q'|q) = \sum_{j \in q} \mathbb{P}[q \rightsquigarrow j;\psi] H(q'|q \rightsquigarrow j) = -\sum_{j \in q} \mathbb{P}[q \rightsquigarrow j;\psi] \sum_{j' \in q'} \mathbb{P}[q' \rightsquigarrow j'|q \rightsquigarrow j] \log \mathbb{P}[q' \rightsquigarrow j'|q \rightsquigarrow j]$. This approach uses the approximately-optimal policy query as a target for finding a good askable query. Concretely, I want the askable query that minimizes the robot's uncertainty about which

response would have been chosen if the desired policy query could have been asked and answered. While the query found this way is not necessarily the optimal query in Q (because there might be no askable query that is a good proxy for asking q^*_Π), it often does well (as is shown empirically later) and avoids extensive computation.

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

Using the query projection method described in Eq. 5.7, the robot finds the start state for the trajectory query based on q^*_Π as the state maximizing the heuristic value:

$$h^{QP}(s) = \mathbb{E}_{q_U \sim U(s)} \sum_{u \in q_U} \mathbb{P}(q_U \rightsquigarrow u; \psi) H(q^*_\Pi | q_U \rightsquigarrow u; \psi) \tag{5.8}$$

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

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

Also, instead of first greedily forming policy queries and then projecting them to the trajectory query space, one could ask why I do not just greedily generate trajectory queries directly, which is what the Expected Belief Change and the Query by Disagreement strategies (Wilson, Fern, and Tadepalli 2012) (that I soon describe) do. The reasons are two-fold. First, since trajectory queries are not decision queries, the greedy construction procedure applied to trajectories does not enjoy the near-optimality guarantee for policy queries, and

thus could be arbitrarily suboptimal. I instead project from the k-ary policy query that the robot provably would want to ask (if it could) to find the k-ary trajectory query that best matches it. The second reason is that our projection technique generalizes to any spaces of "askable queries," while heuristics for greedily constructing trajectory queries do not necessarily generalize.

5.4 Empirical Evaluations

In this section, I will empirically confirm the following.

(1) Optimizing the query selection objective (EPU or $EVOI$) using a greedy construction approach is better than using an existing approach based on sampling. I will show this in Sec. 5.4.1.

(2) Greedily-constructed policy queries can approximate optimal policy queries in performance while being computed more cheaply. I will show this in Sec. 5.4.2.

(3) Good trajectory queries can be found through projection of greedily-constructed policy queries. I now empirically test these claims. I will show this in Sec. 5.4.3.

(4) I will also show the advantage of the greedy construction in a discrete driving domain in Sec. 5.4.4.

I compare the performance of our greedy construction method for policy queries against optimal policy queries in a version of the Rock Sample domain (Smith and Simmons 2004), which I call rock collection, which I have purposely made to be sufficiently small for us to be able to exhaustively compute the optimal policy query. The robot navigates in a 5×30 gridworld, where it starts in the middle of the southern border and its actions are constrained to moving straight north, diagonally north-east, or diagonally northwest. The transition function is noisy: with probability 0.05, the robot reaches one of its north, north-east and north-west states uniformly randomly regardless of the action it takes. When the robot hits the boundary on either side, it will only move north rather than off the domain. The length of the trajectories it can follow is 4. There are m rocks randomly distributed in the grid, and the robot knows that the human's true reward function is one of n equiprobable reward function candidates. The rock collection problem lets us adjust the number of reward candidates n and query choices k, so I can learn more deeply about the strengths and limitations of our projection approach.

To generate the reward function candidates, I partition the m rocks into n subsets, and each reward candidate corresponds to a different partition containing rocks that the human

wants collected (positive reward), while all of the other rocks have rewards of -1. I consider three settings for the positive rewards, illustrated (using smaller grids) in Figure 5.2.

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

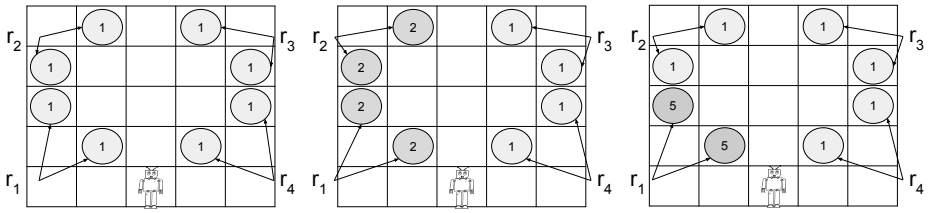
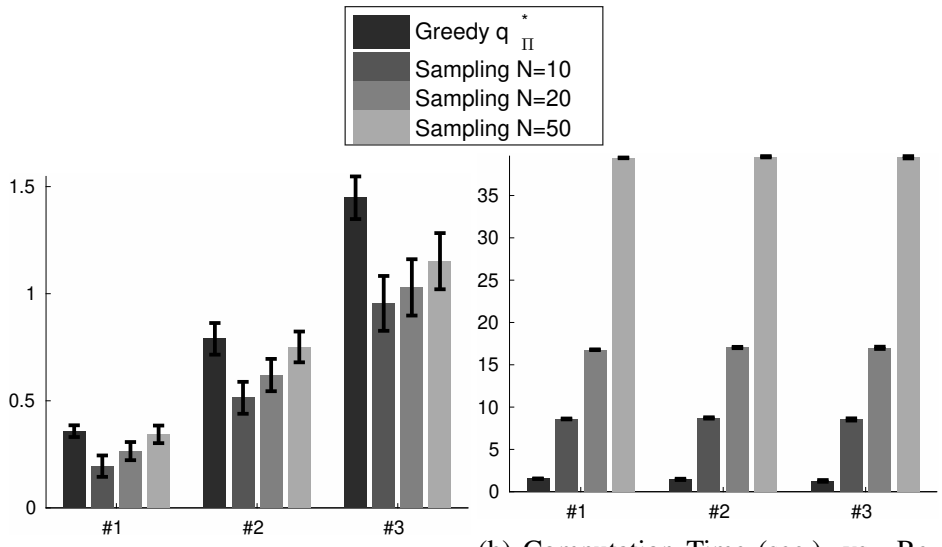


Figure 5.2: Reward settings of the rock collection domain. The locations of rocks are randomly assigned.

5.4.1 Comparison with a Sampling Algorithm

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

Figure 5.3 shows that finding a greedy policy query using my MILP approach is considerably cheaper, computationally, than using the sampling approach for any N considered. Its $EVOI$ is always larger than Sampling even when I sample 50 queries ($N = 50$). This performance gap is caused by the fact that my approach finds a policy query by solving the optimization problem, rather than by sampling policy queries and hoping to find a good one.



(a) $EVOI$ vs. Reward Settings.

(b) Computation Time (sec.) vs. Reward Settings.

Figure 5.3: Comparison of methods for finding policy queries in the rock collection domain. $n = 10, k = 2$.

5.4.2 Comparison with the Optimal Query

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

I report the results in Figures 5.5 and 5.6. Although I consider EPU as the objective so far, I evaluate our performance by $EVOI$, which is always nonnegative. $EVOI$ thus makes it easy to see how much a query helps to improve the robot's performance. The left column shows the case of 5 reward candidates ($n = 5$), and the right column has 10 ($n = 10$). From top to bottom in each figure, I increase the number of responses, k. The locations of rocks are randomly generated and different in different trials. 40 trials are run for each data point. To estimate the heuristic value for one state, I sample trajectories for 20 times to approximate the expectation ($\mathbb{E}_{q_U}$) in the heuristic functions. The error bars represent 95% confidence intervals. The variance of the results is caused by sampling trajectories from policies and the different initial configurations of the domain.

As seen in Figure 5.5, the cases where the number of reward candidates is 5 ($n = 5$)

reinforce that greedy construction can perform well since the gap between the optimal policy query and the greedily-constructed policy query is small. Clearly, optimal q^*_Π has the longest running time (shown in Fig. 5.6). It is much more computationally expensive compared with Greedy q^*_Π when we increase the number of reward candidates ($n = 8$).

5.4.3 Evaluation of Trajectory Queries

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

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

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

$$h^{BC}(s) = \mathbb{E}_{q_U\sim U(s)} \mathbb{E}_{\psi'\sim\Psi(\psi,q_U)} \sum_i |\psi'_i - \psi_i| \tag{5.9}$$

where $U(s)$ is a set of optimal policies of k reward candidates starting from s, which are sampled according to ψ. As in the projection method, q_U is a set of l-length prefixes of

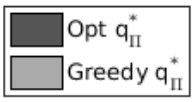
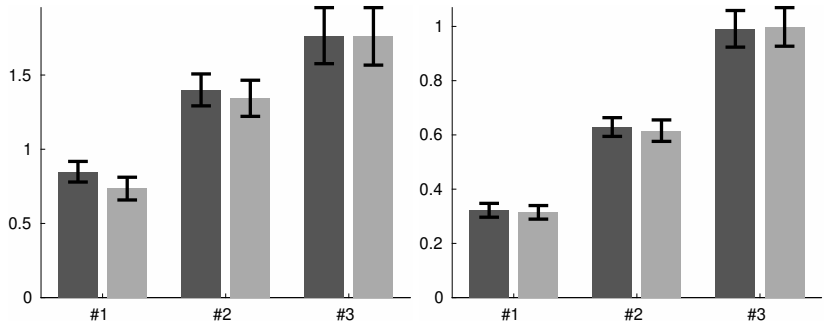
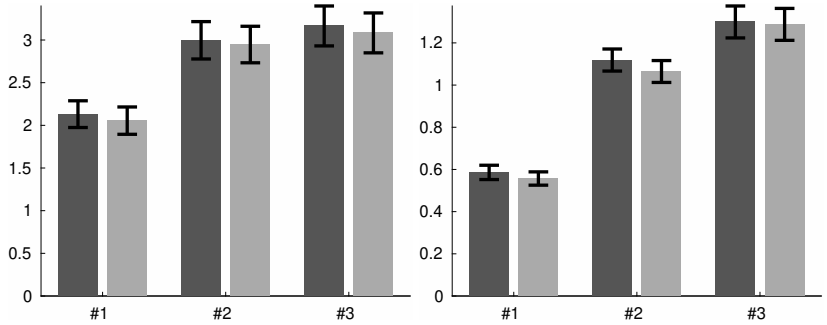


Figure 5.4: Legend for Figures 5.5 and 5.6.



(a) $n = 5, k = 2$. (b) $n = 8, k = 2$.



(c) $n = 5, k = 4$. (d) $n = 8, k = 3$.

$EVOI$ vs. Reward Settings.

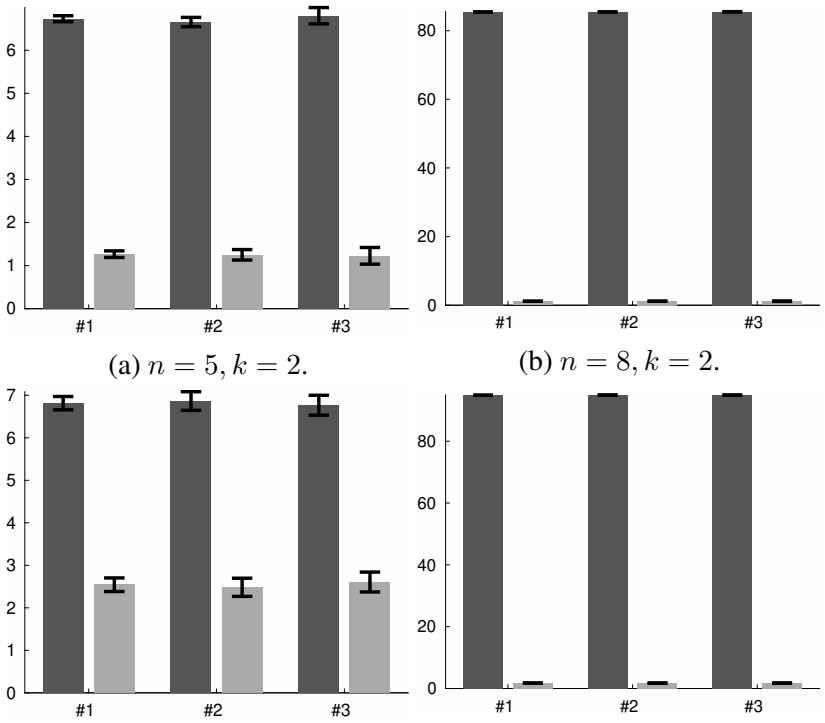
Figure 5.5: $EVOI$ of policy query selection methods in the rock collection domain. Legend is in Figure 5.4.

trajectories starting from s drawn from the policies in $U(s)$. $\Psi(\psi, q_U)$ is the set of posterior beliefs by asking q_U with the prior belief ψ. This heuristic function estimates how much the belief distribution changes.

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

$$h^D(s) = \sum_{\pi, \pi' \in U(s)} \mathbb{E}_{u \sim \pi, u' \sim \pi'} d(u, u') \tag{5.10}$$

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



(a) $n = 5, k = 2$. (b) $n = 8, k = 2$.

(c) $n = 5, k = 4$. (d) $n = 8, k = 3$.

Computation Time (sec.) vs. Reward Settings

Figure 5.6: Computational time of policy query selection methods in the rock collection domain. Legend is in Figure 5.4.

I compare the following algorithms in Figures 5.8 and 5.9. Optimal q^*_Π and greedy q^*_Π in Figures 5.5 and 5.6 are duplicated here for comparison.

1. **Query Projection** finds a trajectory query using h^{QP} in Eq. 5.8 that projects Greedy q^*_Π into the trajectory query space.

2. **Belief Change** finds a trajectory query using h^{BC} in Eq. 5.9.

3. **Disagreement** finds a trajectory query using h^D in Eq. 5.10.

4. **Random Query** uniformly randomly picks a state and generates random trajectories from it.

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

We can see in Figure 5.8 that the query projection method outperforms the other heuristic methods. The reason is that our approach is aware of what the robot should ask given how few choices should be offered. Furthermore, the performance gap grows for reward

settings #2 and #3 because the query projection method is aware of which reward candidates to distinguish from the others, while the other heuristic methods focus on reducing uncertainty of reward beliefs more broadly.

Now I compare the computation time between these methods in Figure 5.9. Query projection is slightly slower than Greedy q^*_Π, since query projection needs to project the policy query it finds to the trajectory query space. Belief Change and Disagreement only need to compute a heuristic function for all states, which are both faster than query projection. Since they are not using the information from the approximately-optimal policy query, their performance is worse than Greedy q^*_Π.

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

5.4.4 Evaluation in Discrete Driving Domain

The previous experiments were in a parameterized domain that I could flexibly modify to test our method under different conditions; I now briefly consider our method's performance in a less manipulable domain. The discrete driving domain I use is based on the driving domain of (Abbeel and Ng 2004). I consider five reward candidates.

- Nice driver, who avoids hitting other cars (reward of -10 for hitting a car).
- Dangerous driver, who avoids hitting other cars, but gets very close to the car in front of him before switching lanes (reward of -10 for hitting a car, but +1 for being right behind another car).
- Nasty driver, who always hits other cars (reward of +1 for hitting a car).
- Left lane nice driver, who is like the nice driver but prefers staying in the left-most lane in the absence of other cars. The driver gets a reward of 0.01 when in the left-most lane.
- Right lane nice driver, who is like the left-lane driver except prefers the right lane.

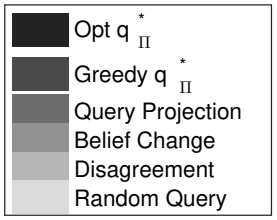
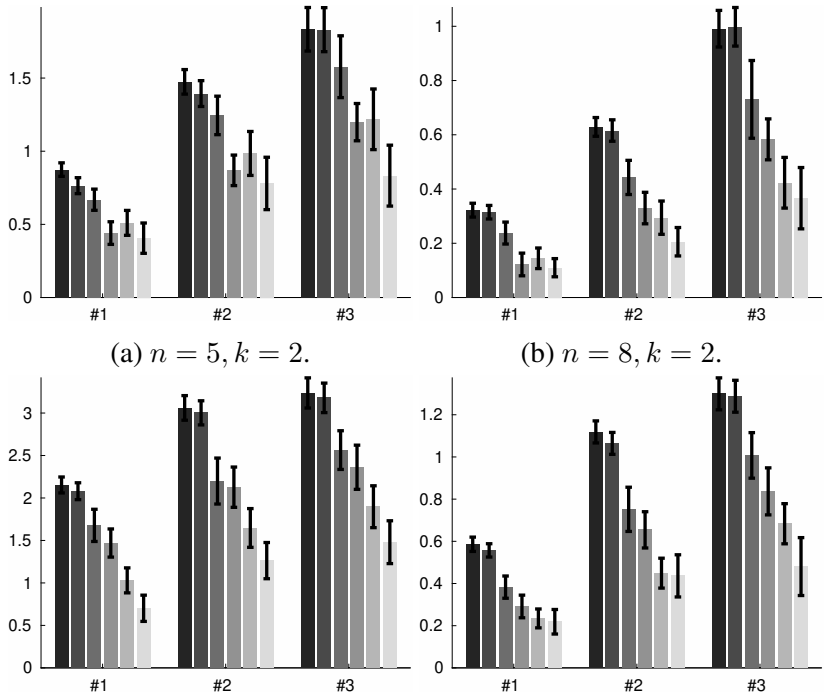


Figure 5.7: Legend for Figures 5.8, 5.9 and 5.11.



(a) $n = 5, k = 2$. (b) $n = 8, k = 2$.

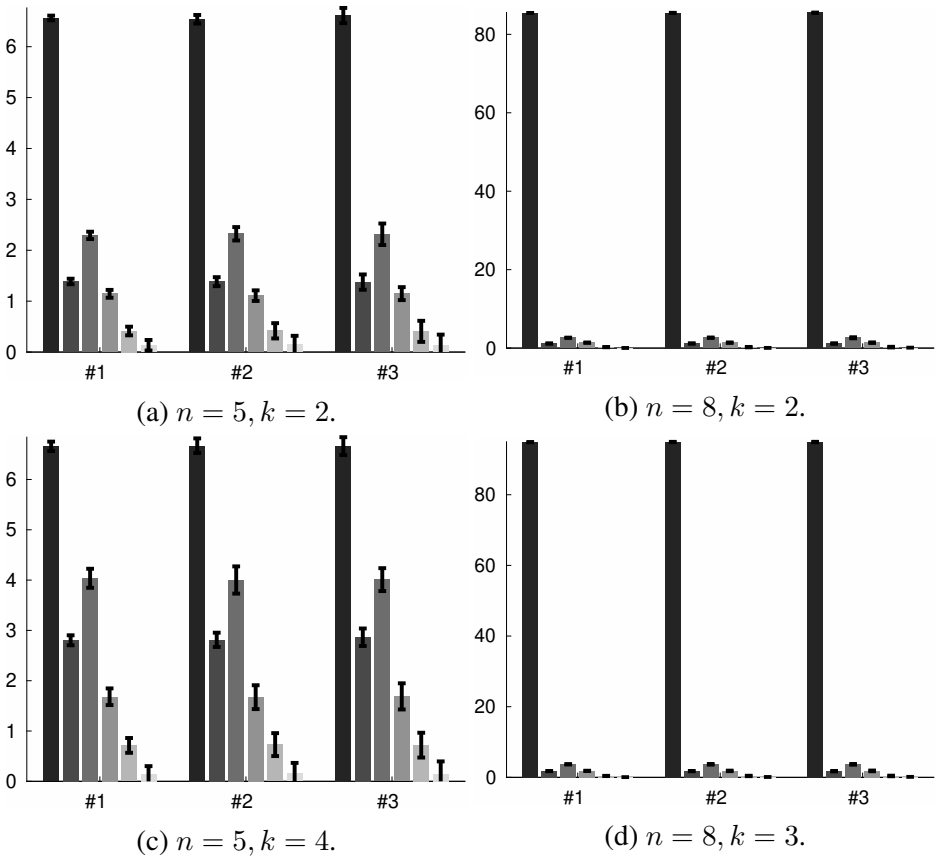
(c) $n = 5, k = 4$. (d) $n = 8, k = 3$.

$EVOI$ vs. Reward Settings.

Figure 5.8: Comparison in the rock collection domain. Legend is in Figure 5.7.

The domain and some of the reward functions are illustrated in Figure 5.10. All of these are from (Abbeel and Ng 2004), except the "dangerous driver" whom I added to increase the variety of the behaviors.

The query projection method has the same performance as the greedily-constructed policy query. The Expected Belief Change heuristic has worse performance when $k = 2$, but it catches up when I increase k. Like Reward Setting #3 in the rock collection problem, distinguishing different reward candidates may yield significantly different EPU than heuristics that focus on reducing overall reward uncertainty. For example, I want to build a binary response query by considering the trajectories in the illustration in Figure 5.10 (bottom right). A good query would select the middle query and either one on the side to decide whether the human wants to hit another car. However, the competing methods do not



(a) $n = 5, k = 2$. (b) $n = 8, k = 2$.

(c) $n = 5, k = 4$. (d) $n = 8, k = 3$.

Computation Time (sec.) vs. Reward Settings

Figure 5.9: Computational time in the rock collection domain. Legend is in Figure 5.7.

discern such distinctions. Query by Disagreement selects the left-most and the right-most trajectory, and Expected Belief Change does not have a strong bias over these trajectories since the posterior beliefs are all changed.

5.5 Conclusion

In this chapter, I consider the setting of query selection under reward uncertainty. I have formulated a method for greedily constructing an approximately-optimal policy query that avoids enumerating all policy choices, and have empirically shown a randomized generated domains (in the rock collection domain in Figure 5.8) that the approach finds queries close to the optimal query found by a brute-force approach.

I also presented an approach for projecting a policy query into a more easily askable space of trajectory queries, using the same idea as in Cohn (2016). I showed empirically that the query found can be better than other trajectory-query selection strategies, particularly when the query should limit the trajectory choices to a small number. The idea of

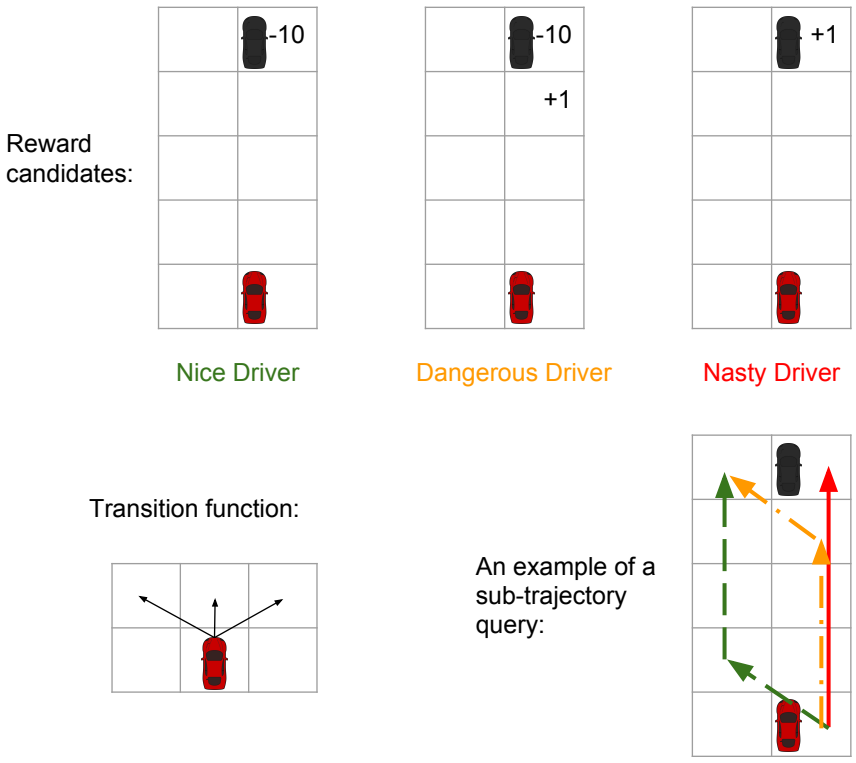


Figure 5.10: The driving domain.

projection can be helpful if we can optimize queries in one space but we can only select a query in a different space.

Regarding future work, currently, I only consider a small number of reward candidates so that I can solve mixed integer linear programming problems with each integer variable corresponding to a reward candidate. Such problems would be slow to solve when we have a lot of reward candidates. It would be of interest to find approximate methods that are suitable when the set of reward candidates is large. Also, I assumed that the human's responses were always correct. The query selection algorithm may need to be modified if the human's response is noisy.

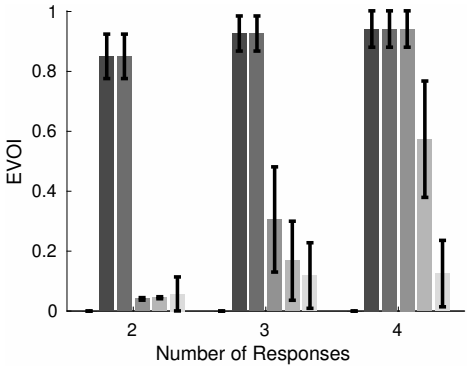


Figure 5.11: Evaluation on the driving domain. The horizontal axis is the number of trajectories in the trajectory query (k). The optimal query q^*_Π cannot be computed.

CHAPTER 6

Efficiently Finding Optimal Queries Under (Only) Safety-Constraint Uncertainty

In the previous chapter, I considered querying under only reward uncertainty. In this chapter, I consider the problem of querying under only safety-uncertainty defined in Sec. 3.2.

When a human user tasks the robot with optimizing a reward function, the robot's policy generally changes other features (e.g., its own position and power level, opening doors, moving furniture, scaring the cat). In any but the most trivial of problems, the robot's optimal policy will change some features of the world in the process. Some of the side-effects in achieving the goal might be fine (and even expected) by the human (e.g., the robot will have to move), but other side-effects might be undesirable/unsafe (e.g., leaving doors open such that the cat can get into the kitchen or out of the house) even though they correspond to faster goal achievement (e.g., leaving doors open allows faster movement between rooms; cleaning the sink is faster by tossing the dirty dishes into the trash compactor rather than washing, drying, and stacking them).

I assume that the human will tell the robot some of the (more obvious) features that can be changed, as well as some that are critical to leave unchanged (e.g., be careful not to knock over the priceless vase). But the human will lack the time, patience, or foresight to explicitly enumerate every feature and tell the robot whether it can or cannot be side-effected. The human may also (probably incorrectly) assume that the robot has common sense, such as about cat behavior and the value of dishes.

So how can the robot execute a **safely**-optimal policy in such a setting? I adopt the baseline conservative assumption that the human is content with the initial values of features whose changeabilities are not specified: if the goal could be achieved without side-effecting any features other than those known to be allowable by the human, then the human is happy with the robot's plan. Thus, any policy that leaves features not explicitly known

to be changeable fixed are considered safe, and the simplest approach towards a safely-optimal policy would be to execute the policy that is optimal when constrained to also be a safe policy. I will assume that such a policy exists in this chapter.

Altogether there are three main contributions in this chapter. 1) I formulate (Section 6.1) an AI safety problem of avoiding negative side-effects in factored MDPs. 2) I show (Section 6.2) how to efficiently identify the set of **relevant** features, i.e., the set of features that could potentially be worth querying the human about. Specifically, a feature is not relevant if being able to change it would never improve the robot's plan. 3) I consider (Section 6.3) a minimax-regret criterion when there is a limit on the number of features the robot can ask about, and the robot finds the minimax-regret query by searching the query space with efficient pruning. Finally, I present some empirical results for our algorithms in a simulated robot navigation task (Section 6.4).

6.1 Problem Definition

For a running example of our problem setting, I consider a simulated robot gridworld-navigation domain inspired by Amodei et al. (2016) and depicted in Figure 6.1. There are doors and carpets and boxes as well as a switch; the robot can open/close a door, move a box, traverse a carpet, and flip the switch at the top right corner off or on. The robot starts in the bottom left corner, doors d1 and d2 start in the open state; the switch starts in the on position. The robot can also move itself to an adjacent location vertically, horizontally, or diagonally. For simplicity, the transition function is assumed to be deterministic.

Suppose that the human specifies the robot's goal as that of turning off the switch in the upper-right corner as quickly as *safely* possible. There are multiple paths the robot could take. The shortest path (π_1) would have the robot traverse the carpet, but the carpet would get dirty and the goal specification does not mention whether such a side-effect is tolerable. The robot could instead enter the room through door d1 and then move box b1 or b2 out of the way (π_2 or π_3 respectively), and then pass through door d2 to go to the switch. However, one or more of the boxes could contain fragile objects and should not be moved; again, this is known to the human but not the robot. Or the robot could enter through door d1 and walk upwards (π_4) around all the boxes to ultimately get to the switch. There are of course many other more circuitous paths, e.g., the robot could move box b3 and then return to its starting location before traversing the carpet to the switch, and so on.

As described in Sec. 3.2, I model the domain as a factored Markov Decision Process (MDP) (Boutilier, Dean, and Hanks 1999). The robot partitions the features into known-to-be-free, known-to-be-locked, and unknown features. Their definitions are repeated here.

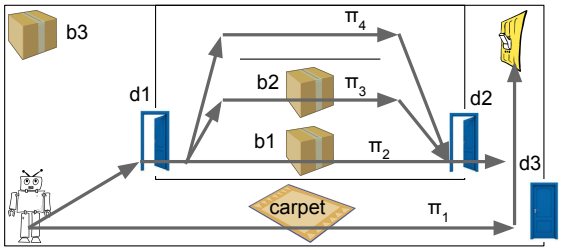


Figure 6.1: The robot navigation domain. The dominating policies are shown as arrows.

- $\Phi_F^{\mathbf{R}}$: The *free-features* (i.e., freely changeable). The robot knows that these features can be changed freely (for example, its location).
- $\Phi_L^{\mathbf{R}}$: The *locked-features*. The robot knows it should never change any of these features.
- $\Phi_?^{\mathbf{R}}$: The *unknown-features*. These are features that the robot doesn't (initially) know whether the human considers freely changeable or locked.

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

6.1.1 Finding Safely-Optimal Policies

To find a safely-optimal policy under a partition of features, I solve the linear programming problem defined in Eq. 2.8, repeated here,

$$\max_x \sum_{s,a} x(s,a) r(s,a) \tag{2.8}$$

$$\text{s.t.} \sum_{a'} x(s',a') = \gamma \sum_{s,a} x(s,a) T(s,a,s') + \alpha(s'), \forall s' \in S \tag{2.9}$$

$$\sum_{a \in A} x(s,a) = 0, \forall s \in S_{\Phi_L^{\mathbf{R}} \cup \Phi_?^{\mathbf{R}}}. \tag{2.10}$$

I assume that the only way for the robot to figure out that an unknown feature is in fact freely changeable is to ask the human a query about that feature. Thus, the focus of the rest of this chapter is on how the robot can ask a good query about a small number, k, of features. In the following sections, I describe algorithms that, under the minimax-regret criterion, ask an optimal k-feature query, first by pruning from $\Phi_?^{\mathbf{R}}$ features that are guaranteed to not be relevant to ask, and then efficiently finding the best k-sized subset of the remaining relevant features to query.

Since I focus on finding a minimax-regret query, I need to compare how much the robot's safe policy is improved after querying. This requires that I make the following assumption in this chapter. I will relax this assumption in Ch. 7.

Assumption 6.1. *The robot knows a safe policy before querying, that is,* $\Pi_{\Phi_L^{\mathbf{R}} \cup \Phi_?^{\mathbf{R}}} \neq \emptyset$.

6.2 Finding All Relevant Unknown-Features

Without querying, recall that to be safe the robot should assume all features in $\Phi_?^{\mathbf{R}}$ should not be changed. If the robot asks a query and finds out that indeed unknown features $\Phi \subseteq \Phi_?^{\mathbf{R}}$ are changeable, then in the linear program I would impose constraints on the features in $\Phi_L^{\mathbf{R}} \cup \{\Phi_?^{\mathbf{R}} \setminus \Phi\}$. Reducing the set of constraints in this way would never reduce the utility achieved by the resulting safely-optimal policy, and will sometimes increase the utility (*a weak monotonicity property*). Throughout, I assume the changeabilities of features are independent, that is, when the robot asks about some features in $\Phi_?^{\mathbf{R}}$, after knowing the human's response, it cannot infer the changeability of other features in $\Phi_?^{\mathbf{R}}$.

Intuitively, when is a feature in $\Phi_?^{\mathbf{R}}$ relevant to the robot's planning of a safely-optimal policy? In the robot navigation domain, if the robot plans to traverse the carpet and take the shortest path to reach the switch (π_1 in Figure 6.1), it will change the state of the carpet (from clean to dirty). If the state of the carpet may not be changed, it may instead plan to enter the room above the carpet (by taking π_2, π_3, or π_4 in Figure 6.1). So the changeability of the carpet is *relevant* since the robot might execute a plan that changes that feature if it is allowed. On the other hand, let's consider the box b3 in the top-left corner. No matter whether other features are changeable or not, a good policy would never consider moving b3. So the changeability of b3 is *irrelevant*. We can see that an unknown-feature is relevant when under some allowed circumstance (where a circumstance is a potential answer to any query), the robot would execute a policy that would side-effect it. Such policies are *dominating policies*, defined formally below.

Definition 6.1. *A* **dominating policy** *is a safely-optimal policy for a circumstance where the unknown features* $\Phi_?^{\mathbf{R}}$ *are partitioned into locked and freely-changeable subsets. I denote the set of dominating policies by*

$$\Gamma = \left\{ \arg\max_{\pi \in \Pi_\Phi} V^\pi : \forall \Phi \subseteq \Phi_?^{\mathbf{R}} \right\}, \tag{6.1}$$

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). I refer to the unknown features side-effected by a policy as *relevant features*:

Definition 6.2. *Let the* **relevant features** *of a policy π be the set of unknown-features changed by π, denoted by $\Phi_{rel}(\pi)$.*

For a set of policies Π, $\Phi_{rel}(\Pi) = \cup_{\pi \in \Pi} \Phi_{rel}(\pi)$.

Finding all relevant features. The set $\Phi_{rel}(\Gamma)$, abbreviated Φ_{rel}, is thus the set of relevant unknown-features for our problem. By definition of dominating policies, a feature that is not relevant for any dominating policy need not be considered for querying by the robot because no matter what the answer by the human is, it would never influence the resulting safely-optimal policy.

To find all relevant features, I first find all dominating policies. Then I can simply check what features are changed by these policies. One way to find all dominating policies is by using the definition in Equation 6.1, but in this way I need to enumerate all exponentially (in $|\Phi_?^{\mathbf{R}}|$) many subsets $\Phi \subseteq \Phi_?^{\mathbf{R}}$ and find each subset's corresponding safely-optimal policy. Instead, in one of our contributions, I develop Algorithm *DomPolicies* (Algorithm 6.1) that incrementally finds dominating policies and can be much more efficient in some cases.

I illustrate Algorithm *DomPolicies*' execution in the robot navigation problem (Figure 6.1). It initializes $\Gamma' = \emptyset$ to which it adds dominating policies as it finds them. Observe that one dominating policy is the safely-optimal policy when I assume all unknown features are changeable, which is π_1 (shown in Figure 6.1). Now $\Gamma' = \{\pi_1\}$. It then looks at what features π_1 changes and finds out that π_1 makes the carpet dirty. It then considers what the robot should do if the carpet is not allowed to be dirty (with any other unknown-features being changeable), i.e., the safely-optimal policy in that circumstance, and finds π_2. Now $\Gamma' = \{\pi_1, \pi_2\}$. Incrementally, it keeps finding safely-optimal policies (which would be dominating policies) by enforcing a subset of $\Phi_{rel}(\Gamma')$ to be unchangeable until there are no more policies to add to Γ'.

Algorithm *DomPolicies* constructs a set of features (Φ'_{rel}) and a set of policies (Γ') simultaneously. I will later establish (in Theorem 6.1) that Γ' is indeed the set of all dominating policies ($\Gamma' = \Gamma$), and $\Phi'_{rel} = \Phi_{rel}$.

In the algorithm, in each iteration, it picks a subset of relevant features, Φ, and, if Φ isn't pruned (as described later), finds the safely-optimal policy with Φ being unchangeable (Line 7). It then adds $\Phi_{rel}(\pi)$, which are the features changed by π, to Φ'_{rel}. It repeats this process until Φ'_{rel} does not grow anymore and all subsets of Φ'_{rel} are considered.

In Line 10, the algorithm solves a linear programming problem with $|S \times A|$ control variables and $|S| + |\Phi|$ constraints to find a constraint-satisfying policy. In the worst case, it needs to solve this linear programming problem $2^{|\Phi_{rel}|}$ times. Note that, while the com-

Algorithm 6.1 *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 $\Phi \subseteq \Phi'_{rel}$ I 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. 3.1
        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', Φ'_{rel}

plexity is exponential, it is exponential in the number of *relevant* features, which as we will see empirically can be much smaller than the number of unknown features ($|\Phi_?^{\mathbf{R}}|$).

The computational efficiency can be improved with the pruning rule shown in Algorithm Pruning (Algorithm 6.2). It takes as input a proposed set of features as well as a history of ordered pairs (proposed features, relevant features for safely-optimal policy). If a subset of the proposed features has already been considered and the resulting policy's relevant features do not intersect with the proposed features, the expensive computation of finding the optimal policy for the proposed features can provably be skipped (i.e., pruned). This is visualized in Figure 6.2, where the proposed feature-set Φ' is a superset of a previously encountered features-set Φ and Φ' does not intersect with the set of relevant features of the safely-optimal policy, π, for Φ. In such a case Algorithm *DomPolicies* can ignore Φ' (because its dominating policy π is already in Γ').

Example 6.1. *In the MDPs of Figure 6.3, the robot's task is to turn off a switch. The rewards are marked on the edges. The features (carpets) that are side-effected when traversing an edge are marked in boxes. In the MDP of Figure 6.3(a), Algorithm DomPolicies only computes the* $n + 1$ *dominating safely-optimal policies for* $\Phi = \emptyset, \{c_1\}, \{c_1, c_2\}, \ldots$. *All other subsets of the unknown features are pruned. In the rather different looking MDP*

Algorithm 6.2 Pruning

```
1: function SATISFY(Φ, β)
2:     for (enforced, relaxed) ∈ β do
3:         if enforced ⊆ Φ and Φ ∩ relaxed = ∅ then
4:             return False
5:         end if
6:     end for
7:     return True
8: end function
```

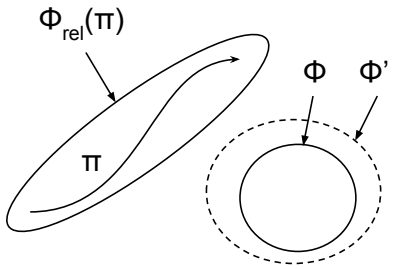


Figure 6.2: Pruning rule illustration.

of Figure 6.3(b), the algorithm finds the 2 dominating policies while only doing policy computations for $\Phi = \emptyset, \{c_1\}, \{c_2\}, \ldots, \{c_n\}$. *All other subsets of unknown features are pruned.*

Thus, in the above example MDPs, Algorithm *DomPolicies*'s complexity is linear in the number of relevant features rather than the worst case of exponential in the number of relevant features. I further confirm that Algorithm *DomPolicies* does find all dominating policies.

Theorem 6.1. *The set of policies,* Γ'*, returned by Algorithm DomPolicies is the set of all dominating policies,* Γ *(Eq. 6.1).*

Proof. ($\Gamma' \subseteq \Gamma$) It is easy to see that any policy in Γ' is a dominating policy, since $\pi \in \Gamma'$ is a safely-optimal policy with a subset of unknown features unchangeable (Line 10 in Algorithm *DomPolicies*).

($\Gamma' \supseteq \Gamma$) Now I want to show that all dominating policies are included in Γ'. Consider a policy $\pi \in \Gamma$ and π is the optimal policy with unknown features C being unchangeable. I denote $C \cap \Phi_{rel}(\Gamma')$ by A and $C \setminus \Phi_{rel}(\Gamma')$ by B (illustrated in Figure 6.4).

Assume $\pi \notin \Gamma'$. Then $B \neq \emptyset$. Otherwise, if $B = \emptyset$ (or equivalently, $A = C$), then C is a subset of $\Phi_{rel}(\Gamma')$ and π would have been added to Γ'. Let π' be the optimal policy with A being unchangeable. Since $A \subseteq \Phi_{rel}(\Gamma')$, I know $\pi' \in \Gamma'$ because the optimal policies

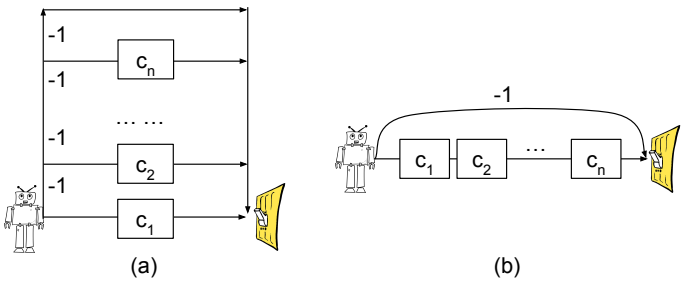


Figure 6.3: Example domains used in text. ($n > 2$)

with any subsets of $\Phi_{rel}(\Gamma')$ being unchangeable are added to Γ'. I also 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 $A \cup B = C$ being unchangeable. So $\pi = \pi'$, which is an element in Γ'. I have a contradiction.

Therefore I have $\Gamma' = \Gamma$. □

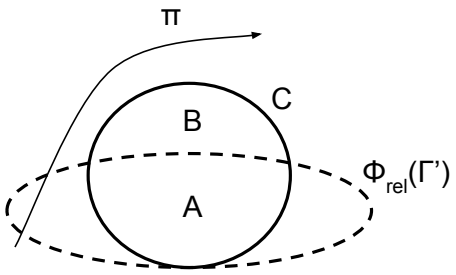


Figure 6.4: Theorem 6.1 illustration.

6.3 Finding Minimax-Regret Queries

The previous section provides a method to find all relevant features that are worth asking, which is the set of features that are changed by some dominating policies. While the set of all relevant features may be much smaller than the set of unknown features, it may still be too large for the robot to query the human about all of the relevant features. So I consider the setting that the robot can ask about a limited and fixed number k of features, and for each the human will reply whether or not it is in $\Phi_F^{\mathbf{H}}$.

Formally, q_Φ is a *k-feature query* where $q_\Phi \subseteq \Phi_?^{\mathbf{R}}$ and $|q_\Phi| = k$. Furthermore, I have proven that, without loss of optimality, I can restrict querying to only relevant features, so $q_\Phi \subseteq \Phi_{rel}$. If $|\Phi_{rel}| \leq k$ then the robot is done: it should just ask about the relevant features. Generally, though, it will need to select k elements to ask from a larger relevant

feature set, and a reasonable criterion is to select the k elements that minimize its possible maximum (worst-case) regret.

I first define the post-response utility when the robot asks query q_Φ and $\Phi_c \subseteq \Phi_?^{\mathbf{R}}$ are actually changeable. This is the value of the safely-optimal policy after the human's response.

$$u(q_\Phi, \Phi_c) = \max_{\pi \in \Pi_{\Phi_?^{\mathbf{R}} \setminus (q_\Phi \cap \Phi_c)}} V^\pi \tag{6.2}$$

Note that the robot can only safely change features it queries about that the human's response indicates are changeable ($q_\Phi \cap \Phi_c$). What would the robot regret if it asks a k-feature query q_Φ rather than a k-feature query q'_Φ? I consider the circumstance where a set of features Φ_c are changeable and under which the difference between the utilities of asking q_Φ and q'_Φ is maximized. I call such difference of utilities the *pairwise maximum regret* of queries q_Φ and q'_Φ, defined below in a similar way to Regan and Boutilier (2010).

$$PMR(q_\Phi, q'_\Phi) = \max_{\Phi_c \subseteq \Phi_?^{\mathbf{R}}} (u(q'_\Phi, \Phi_c) - u(q_\Phi, \Phi_c)) \tag{6.3}$$

The maximum regret of query q_Φ is determined by q'_Φ that maximizes $PMR(q_\Phi, q'_\Phi)$, denoted by MR.

$$MR(q_\Phi) = \max_{q'_\Phi \subseteq \Phi_{rel}, |q'_\Phi| = k} PMR(q_\Phi, q'_\Phi) \tag{6.4}$$

The robot's objective is to find the minimax-regret (k-feature) query:

$$q_\Phi^{MMR} = \arg \min_{q_\Phi \subseteq \Phi_{rel}, |q_\Phi| = k} MR(q_\Phi) \tag{6.5}$$

The rationale of the minimax-regret criterion is as follows. Whenever the robot considers a query q_Φ, there might exists Φ_c such that if Φ_c is the true set of changeable features, there exists a query q'_Φ that is better than q_Φ. The robot focuses on the worst case Φ_c, that is, under such Φ_c, I can find a query q'_Φ and the difference between the utilities of q_Φ and q'_Φ is maximized. This is as if the robot is playing against an adversary. The robot first plays a query q_Φ. The adversary then plays a query q'_Φ and a set of features Φ_c. The adversary wants to maximize the gap between the utilities of q'_Φ and q_Φ under Φ_c, that is, $u(q'_\Phi, \Phi_c) - u(q_\Phi, \Phi_c)$. The robot wants to find a query q_Φ so that such a maximized gap is minimized.

Under the definition of MR, I need to find the maximizing q'_Φ and Φ_c. However, I can simplify the definition in the following way so that I only need to maximize Φ_c. Note that since the adversary has the power to choose both q'_Φ and Φ_c, it wants to make sure that $\Phi_c \subseteq q'_\Phi$, which means that it does not want the features not in q'_Φ to be changeable. With

this observation, I can rewrite MR in the following way.

$$MR(q_\Phi) = \max_{\Phi_c \subseteq \Phi_{rel}, |\Phi_c| \leq k} \left(\max_{\pi' \in \Pi_{\Phi_?^{\mathbf{R}} \setminus \Phi_c}} V^{\pi'} - \max_{\pi \in \Pi_{\Phi_?^{\mathbf{R}} \setminus \{q_\Phi \cap \Phi_c\}}} V^{\pi} \right) \tag{6.6}$$

$$= \max_{\pi' \in \Gamma, \Phi_{rel}(\pi') \leq k} \left(V^{\pi'} - \max_{\pi \in \Pi_{\Phi_?^{\mathbf{R}} \setminus \{q_\Phi \cap \Phi_{rel}(\pi')\}}} V^{\pi} \right) \tag{6.7}$$

This observation also provides a way to compute MR more efficiently. If I compute the maximum regret of each query (per Eq. 6.4), the algorithm needs to enumerate every $\Phi_c \subseteq \Phi_{rel}$. I observe in Eq. 6.7 that I can enumerate dominating policies, which in general will be fewer than the cardinality of the powerset of the relevant features (although in the worst case $|\Gamma| = 2^{|\Phi_{rel}|}$).

I call the π' in Eq. 6.7 selected by the adversary the *adversarial policy* when the robot asks query q_Φ, denoted by $\pi_{q_\Phi}^{MR}$.

$$\pi_{q_\Phi}^{MR} = \arg \max_{\pi' : \pi' \in \Gamma, \Phi_{rel}(\pi') \leq k} \left(V^{\pi'} - \max_{\pi \in \Pi_{\Phi_?^{\mathbf{R}} \setminus \{q_\Phi \cap \Phi_c\}}} V^{\pi} \right) \tag{6.8}$$

While Eq. 6.7 can significantly speed up the maximum-regret computation for a query relative to Eq. 6.4, the robot still faces the need to do this computation for every possible query (Eq. 6.5) of size k. As another contribution, I further improve the efficiency in the following ways. First, I may not need to consider all relevant features if I can only ask about k of them. If a subset of relevant features satisfies the condition in Theorem 6.2, which I call a set of *sufficient features*, then I know that the minimax-regret k-feature query constituted by features in that subset is the globally minimax-regret k-feature query. That is to say, I lose nothing if I ignore the relevant features that are not in the set of sufficient features. Second, it is possible to safely eliminate some queries which are known to be not better than the queries I have evaluated. I introduce a pruning rule that I call *query dominance* in Theorem 6.3.

The following theorem shows that if I can find any subset Φ of Φ_{rel} such that no matter which k-feature subset of Φ is posed as a query, 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 6.2. (Sufficient Feature Set) *For any set of* $\geq k$ *features* Φ*, if for all* $q_\Phi \subseteq \Phi, |q_\Phi| = k$*, I have* $\Phi_{rel}(\pi_{q_\Phi}^{MR}) \subseteq \Phi$*, then* $\min_{q_\Phi \subseteq \Phi, |q_\Phi| = k} MR(q_\Phi) = \min_{q_\Phi \subseteq \Phi_{rel}, |q_\Phi| = k} MR(q_\Phi)$.

Proof. Let's assume Φ described above does not contain a minimax-regret query. I find a k-feature query $q_\Phi \subseteq \Phi$, such that $\Phi \cap q_\Phi^{MMR} \subseteq q_\Phi$, that is, q_Φ contains all features in q_Φ^{MMR} that are also in Φ. Such a q_Φ can be found since $|\Phi \cap q_\Phi^{MMR}| < k$.

Let $A = q_\Phi^{MMR} \backslash q_\Phi$. As illustrated in Figure 6.5 (Right), I claim that $\Phi_{rel}(\pi_{\Phi_q}^{MR}) \cap A = \emptyset$. Otherwise, according to the property of Φ stated in the theorem, some features in A should have been added to Φ. Let $X = \Phi_{rel}(\pi_{\Phi_q}^{MR}) \cap q_\Phi \setminus q_\Phi^{MMR}$, $Y = \Phi_{rel}(\pi_{\Phi_q}^{MR}) \cap q_\Phi^{MMR}$.

$$
\begin{aligned}
MR(q_\Phi) &= V^{\pi_{\Phi_q}^{MR}} - \max_{\pi' \in \Pi_{\Phi_?^{\mathbf{R}} \backslash (X \cup Y)}} V^{\pi'} \\
&\leq V^{\pi_{\Phi_q}^{MR}} - \max_{\pi' \in \Pi_{\Phi_?^{\mathbf{R}} \backslash Y}} V^{\pi'} \;\leq\; MR(q_\Phi^{MMR})
\end{aligned}
$$

which means $MR(q_\Phi) = MR(q_\Phi^{MMR})$ since q_Φ^{MMR} is a minimax-regret query. So q_Φ is also a minimax-regret k-feature query and $q_\Phi \subseteq \Phi$. This is contradictory to the assumption I made at the start. Therefore Φ contains a minimax-regret k-feature query. □

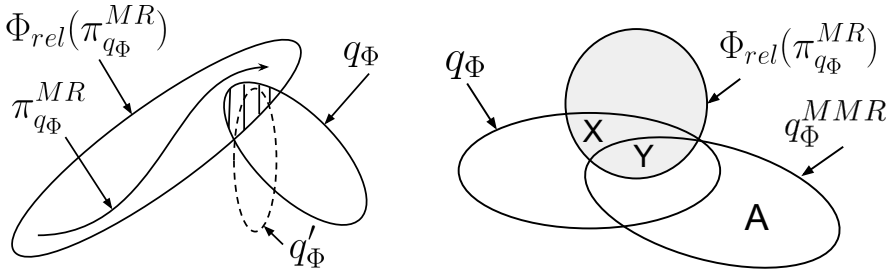


Figure 6.5: (Left) Illustration of the set of features the robot can change, indicated by the shaded area. (Right) Illustration of the proof of Theorem 6.2.

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 6.3. (Query Dominance) *For any pair of queries q_Φ and q'_Φ, if $q'_\Phi \cap \Phi_{rel}(\pi_{q_\Phi}^{MR}) \subseteq q_\Phi \cap \Phi_{rel}(\pi_{q_\Phi}^{MR})$, then $MR(q'_\Phi) \geq MR(q_\Phi)$.*

Proof. Observe that

$$
\begin{aligned}
MR(q'_\Phi) &\geq V^{\pi_{q_\Phi}^{MR}} - \max_{\pi' \in \Pi_{\Phi_?^{\mathbf{R}} \backslash (q'_\Phi \cap \Phi_{rel}(\pi_{q_\Phi}^{MR}))}} V^{\pi'} \\
&\geq V^{\pi_{q_\Phi}^{MR}} - \max_{\pi' \in \Pi_{\Phi_?^{\mathbf{R}} \backslash (q_\Phi \cap \Phi_{rel}(\pi_{q_\Phi}^{MR}))}} V^{\pi'} \;=\; MR(q_\Phi)
\end{aligned}
$$

where the first inequality follows from Eq. 6.7 and the second inequality follows from the first because $q'_\Phi \cap \Phi_{rel}(\pi^{MR}_{q_\Phi}) \subseteq q_\Phi \cap \Phi_{rel}(\pi)^{MR}_{q_\Phi}$, and so $\Pi_{\Phi^{\mathbf{R}}_?\setminus(q'_\Phi \cap \Phi_{rel}(\pi^{MR}_{q_\Phi}))} \subseteq \Pi_{\Phi^{\mathbf{R}}_?\setminus(q_\Phi \cap \Phi_{rel}(\pi^{MR}_{q_\Phi}))}$. □

A query that satisfies such a dominance condition is illustrated in Figure 6.5 (Left) as q'_Φ. If the condition stated in this theorem holds, I will know for certain that $MR(q'_\Phi)$ is not smaller than $MR(q_\Phi)$. I denote the condition $q'_\Phi \cap \Phi_{rel}(\pi^{MR}_{q_\Phi}) \subseteq q_\Phi \cap \Phi_{rel}(\pi^{MR}_{q_\Phi})$ by $dominance(q_\Phi, q'_\Phi)$. To compute dominance, I only need to store $\Phi_{rel}(\pi^{MR}_{q_\Phi})$ for all q_Φ I have considered.

Algorithm *MMRQ-k* (Algorithm 6.3) 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 in the relevant feature set.

Algorithm 6.3 *MMRQ-k*

1: $q_\Phi \leftarrow$ an initial k-feature query
2: $checked \leftarrow \emptyset$ ▷ The set of queries I have examined so far.
3: $evaluated \leftarrow \emptyset$ ▷ The set of queries for which I have computed maximum regrets.
4: $\Phi_{suf} \leftarrow q_\Phi$
5: $agenda \leftarrow \{q_\Phi\}$
6: **while** $agenda \neq \emptyset$ **do** ▷ By Theorem 6.2, I can stop evaluating queries when $agenda$ is empty.
7: $q_\Phi \leftarrow$ an element from $agenda$
8: **if** $\neg\exists_{q'_\Phi \in evaluated} dominance(q'_\Phi, q_\Phi)$ **then** ▷ By Theorem 6.3, I can skip q_Φ if it is dominated by another query.
9: Compute $MR(q_\Phi)$ and $\pi^{MR}_{q_\Phi}$
10: $\Phi_{suf} \leftarrow \Phi_{suf} \cup \Phi_{rel}(\pi^{MR}_{q_\Phi})$
11: $evaluated \leftarrow evaluated \cup \{q_\Phi\}$
12: **end if**
13: $checked \leftarrow checked \cup \{q_\Phi\}$
14: $agenda \leftarrow all\_k\_subsets\_of(\Phi_{suf}) \setminus checked$
15: **end while**
16: return $\arg\min_{q_\Phi \in evaluated} MR(q_\Phi)$

Intuitively, the algorithm keeps augmenting the set of features Φ_{suf}, which contain the features in the queries I 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 I have not yet evaluated. According to Theorem 6.2, I can terminate the algorithm when $agenda$ is empty (Line 6). I also use Theorem 6.3 to filter out queries that I know are not better than the ones I have found already (Line 8). Note that an initial Φ_{suf} needs

to be chosen, which can be arbitrary. I initialize q_Φ with the chain of adversaries heuristic described in Algorithm *CoA*, which I will describe in Section 6.4.

The following example shows how Algorithm *MMRQ-k* can prune suboptimal queries and thus gain efficiency.

Example 6.2. *Let's consider finding the minimax-regret 2-feature query in Figure 6.3 (a), which should be* $\{c_1, c_2\}$.

If the robot considers a query that does not include c_1*, the adversarial policy would change* c_1*, and* c_1 *is added to* Φ_{suf}*. If the robot considers a query that includes* c_1 *but does not include* c_2*, the adversarial policy would change* c_2*, and* c_2 *is added to* Φ_{suf}*. When the robot asks* $\{c_1, c_2\}$*, the adversarial policy changes* c_3*, so* c_3 *is added to* Φ_{suf}*.*

With $\{c_1, c_2, c_3\} \subseteq \Phi_{suf}$*, we can see that the condition in Theorem 6.2 holds and I do not need to consider other features. The minimax-regret query constituted by features in* Φ_{suf} *is* $\{c_1, c_2\}$.

Next is an example when our Algorithm *MMRQ-k* is unable to gain efficiency over the worst-case.

Example 6.3. *I consider finding the minimax-regret 2-feature query in Figure 6.3 (b). I can verify that no queries can be eliminated and I need to consider all queries (*$\binom{n}{2}$*).*

6.4 Empirical Evaluations

I 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. I 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 $q_0 = \emptyset$ and improves this query by iteratively computing:

$$\tilde{\pi} \leftarrow \arg \max_{\pi' \in \Gamma : |\Phi_{rel}(\pi') \cup q_i| \leq k} (V^{\pi'} - \max_{\pi \in \Pi_{\Phi^{\mathbf{R}}_?} \setminus \{q_i \cap \Phi_{rel}(\pi')\}} V^{\pi}) \tag{6.9}$$

$$q_{i+1} \leftarrow q_i \cup \Phi_{rel}(\tilde{\pi}). \tag{6.10}$$

The algorithm stops when $|q_{i+1}| = k$ or $q_{i+1} = q_i$. 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.

Example 6.4. *in Figure 6.6, 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* $\{c_1, c_3\}$.

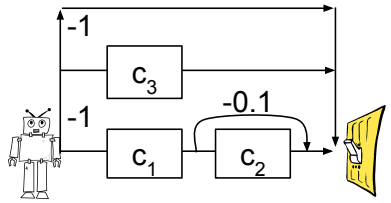


Figure 6.6: Example domain illustrating that *CoA* does not always find the minimax-regret query.

I 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 (or asking dummy queries).

I evaluate the quality of the queries found by these algorithms and also the algorithms' computation time. Although I can present MR of these queries, I report the *normalized* MR to capture the relative performance compared to the best and the worst possible queries. The normalized MR of a query q_Φ is defined as $\frac{MR(q_\Phi)-MR(q_\Phi^{MMR})}{MR(q_\Phi^\perp)-MR(q_\Phi^{MMR})}$, where $q_\Phi^\perp$ is a dummy query, which is a query that contains k dummy features (that are not present in the domain). The normalized MR of a query is 0 when it is as good as the minimax-regret query and 1 when it is as bad as asking a dummy query.

6.4.1 Robot Navigation

As illustrated in Figure 6.7, 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 $\Phi_?^{\mathbf{R}}$. The robot knows that its own location and the state of the switch are in $\Phi_F^{\mathbf{R}}$. Since *MMRQ-k* attempts to improve on an existing safe policy, the left column and the top row never have carpets to ensure there is 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. I report results on 1500 trials. The only difference between trials are the locations of carpets, which are uniformly randomly placed.

In Figure 6.9, I evaluate all the query selection algorithms for different k values, except that I can only afford to run the brute-force method for small k values. First, I compare the brute force method to our *MMRQ-k* algorithm. In Figure 6.9 (left), I empirically confirm that for k values I have evaluated, *MMRQ-k* finds a minimax regret query, matching Brute Force performance (the green and the blue lines on the bottom overlap). In Figure 6.9 (right), the brute force scales poorly as k grows. It becomes computationally intractable when $k = 4$. *MMRQ-k*, benefiting from Theorems 6.2 and 6.3, is more computationally efficient. As we expect, *MMRQ-k* is more computationally expensive than other easy-to-compute heuristics (CoA and Random, in the right figure), but *MMRQ-k* finds queries of better quality than other baseline heuristics.

I then want to see if and when *MMRQ-k* outperforms other candidate algorithms. In Figure 6.9 (left), 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. Random (rel. feat.) and random unsurprisingly have worse performance than *MMRQ-k* and *CoA*. When k is large, approaching the number of unknown features, being selective is less important. As k approaches 10, all methods except no query find good queries (close to 0 normalized maximum regret).

I also consider how $|\Phi_{rel}|$ affects the performance. In Figure 6.10, when $|\Phi_{rel}|$ is smaller than k ($|\Phi_{rel}| \leq 2$ in the left figure and $|\Phi_{rel}| \leq 4$ in the right figure), 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: We see that the green, red, solid purple curves all overlap. When $|\Phi_{rel}|$ is slightly larger than k (when $|\Phi_{rel}|$ is slightly larger than 2 in the left figure and when $|\Phi_{rel}|$ is slightly larger than 4 in the right figure), *CoA* unsurprisingly finds queries close to the minimax-regret queries. The red curve is closest to the green curve. When $|\Phi_{rel}|$ is much larger than k and approaches

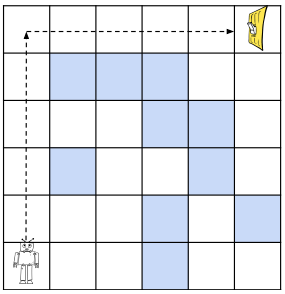


Figure 6.7: Office navigation and legend for following figures.

10, query selection needs to become more selective, and thus the gap between *MMRQ-k* and other algorithms is larger: All the curves except the green curve are going up when k reaches 10. The error bars are larger for small $|\Phi_{rel}|$ since more rarely are only very few features relevant. In summary, *MMRQ-k*'s benefits increase with the opportunities to be selective (larger $\binom{|\Phi_{rel}|}{k}$).

6.5 Discussion and Conclusion

This chapter tackles the problem of how a robot should intelligently query a human user about what features can or cannot be safely side-effected. I considered a minimax-regret objective in this chapter so that I can exploit the existing ideas from the literature about dominating policies, minimax regret, and pruning. Theses ideas are used in a novel way in designing the algorithm (Algorithm 6.1 and Algorithm 6.3) to find better policies while provably maintaining safe optimality. Although it is difficult to prove that such algorithms have less computational complexity, I show empirically that these methods can be much more computationally efficient than brute force methods.

The algorithm in this chapter may still be impractical in large domains. We can see that to find an optimal query, we need to compute the safely-optimal policies under different partitions of features. It can be computationally expensive to evaluate all k-subsets of features even using the pruning techniques. 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 as they just become less likely to help. Once the robot updates its belief after querying, it still guarantees to follow a safe policy by making sure its policy satisfies the updated safety constraints.

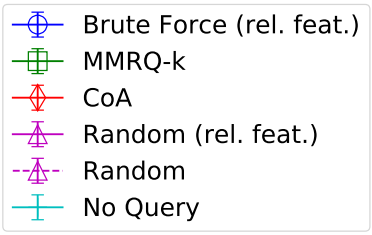


Figure 6.8: The legend for the following figures.

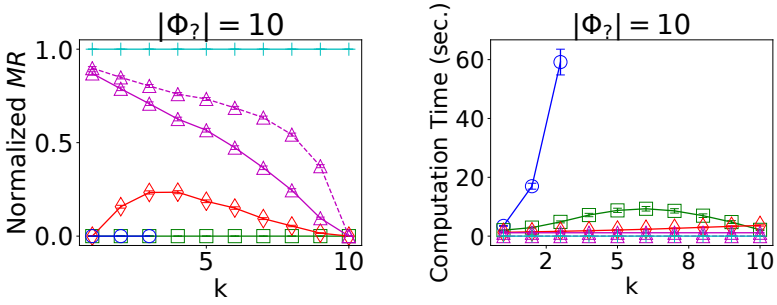


Figure 6.9: Normalized maximum MR vs. k. $|\Phi_?| = 10$. Brute force computation time is only shown for $k = 0, 1, 2$.

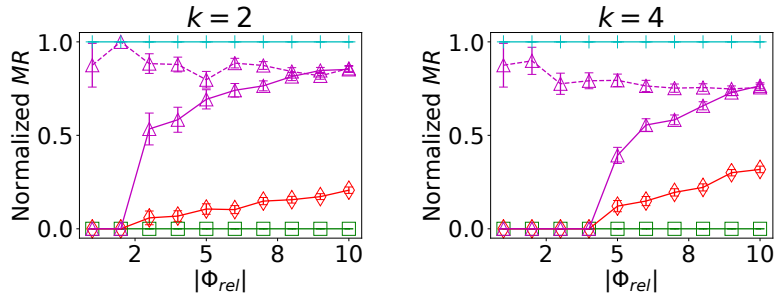


Figure 6.10: Normalized MR vs. the number of relevant features. $|\Phi_?| = 10$ and $k = 2, 4$.

Also, I considered a minimax-regret objective without a Bayesian prior. If the robot has a Bayesian prior over the changeability of features, that may require a different approach since the algorithms in this chapter exploit the property of maximum regret. The query selection algorithms in the next chapter assume a Bayesian prior. An interesting future direction is to adapt the query selection algorithm in the next chapter to the setting of improving safe policies in this chapter.

Lastly, I assumed the existence of initial safe policies. This may not be the case in reality since there may not exist a known initial safe policy under the robot's initial partition of features. I will relax these assumptions in the following chapter (Ch. 7). I also assumed that the human only prefers a feature to be either locked or free. The human may have more preferences than this. For example, the human allows a feature to be changed as long as it is changed back at the end of the episode. While we have done a small investigation on this

extension in Sec. 6 in Zhang, Durfee, and Singh (2018), a more thorough treatment is left for future work.

CHAPTER 7

Querying to Find an Initial Safe Policy

In the previous chapter, I assumed that the robot knows an initial safe policy. In reality, the robot may not initially know any safe policy. I consider this setting in this chapter.

For example, Figure 7.1 shows a robot navigation domain modified from the previous chapter. The only difference is that d2 is now closed. 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 I 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 after 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*.

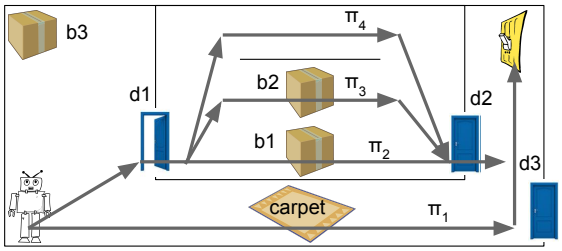


Figure 7.1: The robot navigation domain.

Since no initial safe policy is known, the algorithm in the previous chapter cannot be applied. This chapter's contribution thus is to answer the question of how the robot should efficiently 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 algorithm to pose queries to improve the policy.

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

7.1 Problem Statement

Identical to Ch. 6, I model the domain as a factored Markov decision process (Boutilier, Dean, and Hanks 1999). The robot partitions the features into known-to-be-free features, known-to-be-locked features and unknown features. Like Ch. 6, I require that *the robot should not change any features in* $\Phi_L^{\mathbf{R}}$ *or* $\Phi_?^{\mathbf{R}}$ (**Requirement 1**). That is, it can only change features that are known to be free (in $\Phi_F^{\mathbf{R}}$).

Only allowing the robot to change free features may not lead to satisfactory policies. For example, in Figure 7.1, the robot can simply stay in place to avoid changing any unknown features. Though safe, this behavior receives no positive rewards, and I do not want the robot to follow such a trivially safe policy. Hence, I 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 δ_G* (**Requirement 2**). The robot knows both S_G and δ_G. Note that enforcing the occupancy on the goal states is not equivalent to guaranteeing the probability of reaching the goal states. I consider occupancy constraints so it is easy to impose the constraint in the following linear programming program (Eq. 7.1). We can impose the probability of reaching the goal at a time step, but that needs to add the current time step to the state representation, which may dramatically increase the state space. I will leave other types of constraints on reaching the goal to future work.

I 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 7.1 where the carpet, boxes, and doors are unknown features. Although k-feature queries are useful to find a better safe policy in Ch. 6, they may be not the best query language in this problem. The goal is to reach either outcome (finding a safe policy or proving that none exists). After posing a k-feature query, the robot may still not be able to reach either outcome. So I allow the robot to pose *iterative queries*, following the convention in the literature (Akrour, Schoenauer,

and Sebag 2012; Le et al. 2018; Regan and Boutilier 2009). It can query about a single ($k = 1$) unknown feature $\phi \in \Phi^{\mathbf{R}}_?$, then receive the human's response ($\phi \in \Phi^{\mathbf{H}}_F$ or $\phi \in \Phi^{\mathbf{H}}_L$), and move the queried feature from $\Phi^{\mathbf{R}}_?$ to $\Phi^{\mathbf{R}}_F$ or $\Phi^{\mathbf{R}}_L$. As needed, it can then repeat this process.

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 7.1, 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 7.1, 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 $\Phi^{\mathbf{H}}_F$ and $\Phi^{\mathbf{H}}_L$. 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 (Mindermann et al. 2018; Ramachandran and Amir 2007), I assume that the robot has a prior over the human's preferences. It knows the probability of any unknown feature $\phi \in \Phi^{\mathbf{R}}_?$ being free, denoted by $p_F(\phi)$. The probability that ϕ 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 $\bot$).*

Note that the Bayesian assumption is new in this chapter compared with Ch. 6. I find that without a Bayesian prior, it would be more challenging to find a good querying policy in this query selection problem. In some domains, without a Bayesian prior, the *worst case* number of queries posed is the same no matter in what order the queries are posed. For example, in Figure 7.1, in the worst case, the robot has to query about d2 and the carpet to decide if a safe policy exists. In that case, there is no difference between a good or bad querying policy.

I 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. My contribution is to identify and exploit set-cover subproblems within the problem to design superior heuristics for a greedy approach to query selection.

Finding safely-optimal policies. The following LP problem finds the occupancy measure of the safely-optimal policy, This is almost the same as Eq. 3.1. The only difference is that I encode Requirement 2 as a constraint (Eq. 7.3).

$$\max_x \sum_{s,a} x(s,a) r(s,a) \tag{7.1}$$

$$\text{s.t.} \sum_{a'} x(s',a') = \gamma \sum_{s,a} x(s,a) T(s,a,s') + 1_{[s'=s_0]}, \forall s' \in S \tag{7.2}$$

$$\sum_{s \in S_G, a \in A} x(s,a) \geq \delta_G \tag{7.3}$$

$$\sum_{s \in S_{\overline{\phi}}, a \in A} x(s,a) = 0, \phi \in \Phi_L^{\mathbf{R}} \cup \Phi_?^{\mathbf{R}} \tag{7.4}$$

7.2 Querying to Find a Safe Policy

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

I use $(\Phi_L^{\mathbf{R}}, \Phi_F^{\mathbf{R}}, \Phi_?^{\mathbf{R}})$ to refer to the robot's partition of features. When I want to be precise about the updated partition of features during the query process, I use ψ to denote the current partition of features: $\psi = (\Phi_L^{\psi}, \Phi_F^{\psi}, \Phi_?^{\psi})$. Let ψ_0 be the initial partition. Clearly, $\Phi_L^{\psi} \supseteq \Phi_L^{\psi_0}$; $\Phi_F^{\psi} \supseteq \Phi_F^{\psi_0}$ for all ψ reached by any query policy. I denote by π_q a **query policy**. Given a partition ψ, $\pi_q(\psi)$ 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 7.2 (c)). Suppose I 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 I would prefer not exhaustively evaluate all possible query policies to find the optimal one.

7.2.1 Myopic Query Selection

Although finding the provably-optimal query policy is intractable, I can take advantage of the fact that I do not need to find a complete query policy before posing a query. I can instead myopically select a feature to query about, and then decide on the next feature (if any) to query about depending on the human's response. Algorithm 7.1 gives the skeleton

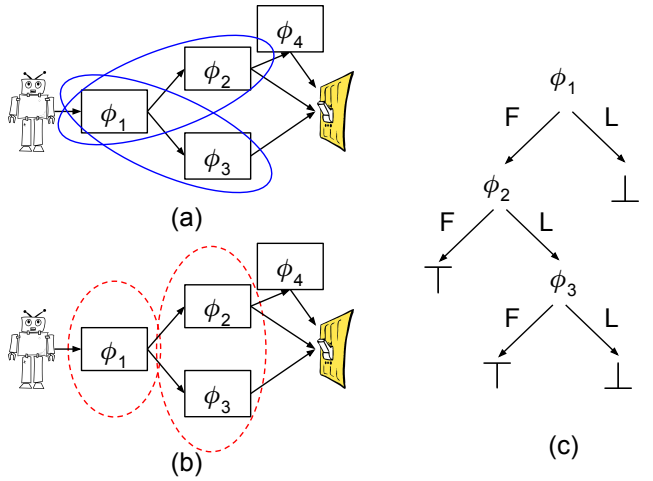


Figure 7.2: Example with 4 unknown features (3 of which are relevant). In (a) I show relevant features of dominating policies; In (b) I show IISs. In (c) is an optimal query policy for the setting in Example 7.1. 'F' means the queried feature is free and 'L' means the queried feature is locked.

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.

I can use Eq. 7.1 in Line 13 to update the truth values of $\top$ and $\bot$. If Eq. 7.1 has a feasible solution under the new partition of features, then the robot finds a safe policy. If Eq. 7.1 does not have a solution even if $\Phi_?^\psi$ 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: $\arg\max_{\pi\in\Gamma}\prod_{\phi\in\Phi_{rel}(\pi)} p_F(\phi)$. Then it queries about its relevant (unknown) feature that has the largest value of $p_F(\cdot)$. I refer to this heuristic as **most-likely-policy**. I see in the following example that it does not always find the optimal query policy.

Example 7.1. *In Figure 7.2, let $p_F(\phi_1) = 0.5$, $p_F(\phi_2) = p_F(\phi_3) = 0.6$. Aiming to find a safe policy, the most-likely-policy heuristic would first query about ϕ_2 or ϕ_3 since its p_F value is higher. However, it is easy to verify that an optimal query policy would start with querying about ϕ_1 first. The optimal query policy, illustrated in Figure 7.2 (c), in expectation asks about 1.7 features, while using the most-likely-policy asks about 2.24.*

In more detail, the most-likely-policy heuristic asks about ϕ_2 (or ϕ_3) first. Depending

Algorithm 7.1 Myopic Query Selection

given query selection criterion h, initial partitions $\Phi_F^{\psi_0}, \Phi_L^{\psi_0}, \Phi_?^{\psi_0}$
$\psi \leftarrow \psi_0$
while not ($\top$ or $\bot$) **do**
    choose ϕ_q according to h based on partition ψ
    query the human about ϕ_q
    $\Phi_?^{\psi'} \leftarrow \Phi_?^{\psi} \setminus \{\phi_q\}$
    if ϕ_q is free **then**
        $\Phi_F^{\psi'} \leftarrow \Phi_F^{\psi} \cup \{\phi_q\}$
    else ▷ ϕ_q is locked
        $\Phi_L^{\psi'} \leftarrow \Phi_L^{\psi} \cup \{\phi_q\}$
    end if
    $\psi \leftarrow \psi'$
    update $\top$ or $\bot$ based on the new partition
end while
if $\top$ is true **then**
    return safely-optimal policy under ψ by solving Eq. 7.1
else ▷ $\bot$ is true
    return "No safe policy exists."
end if

on the response, it will ask about ϕ_1 (if ϕ_2 is free) or ϕ_3 (if ϕ_2 is locked). When ϕ_2 is locked and ϕ_3 is free (this happens with probability $0.4 * 0.6$), it will ask about ϕ_1. So the most-likely-policy heuristic asks about $1 + 1 + 0.4 * 0.6 * 1 = 2.24$ features in expectation.

The optimal query asks about ϕ_1 first, and then about ϕ_2 when ϕ_1 is free (with probability 0.5), and then about ϕ_3 when ϕ_1 is free and ϕ_2 is locked (with probability $0.5 * 0.4$). So it asks about $1 + 0.5 * 1 + 0.5 * 0.4 * 1 = 1.7$ features in expectation.

I will describe other heuristics in Sec. 7.3 as possible candidates, but in the following subsections, I first introduce our main contribution of this chapter: I reveal a set-cover structure in the problem and exploit the structural properties to select better queries.

7.2.2 Set Cover Formulation

In this subsection, I 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, $\top \iff \exists \pi \in \Gamma, \Phi_F^{\mathbf{R}} \supseteq \Phi_{rel}(\pi)$. This is not a set cover problem and does not give us insights on which

feature to query about. But I 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, I know some of their relevant features are locked, then it is impossible to find a safe policy. Formally, let $\mathcal{S}_{rel} = \{\Phi_{rel}(\pi) : \pi \in \Gamma\}$. Then

$$\bot \iff \forall \Phi \in \mathcal{S}_{rel}, \Phi_L^{\mathbf{R}} \cap \Phi \neq \emptyset. \tag{7.5}$$

Example 7.2. *In Figure 7.2 (a), $\mathcal{S}_{rel} = \{\{\phi_1, \phi_2\}, \{\phi_1, \phi_3\}\}$. If the robot knows that ϕ_1 is a locked feature, or ϕ_2 and ϕ_3 are both locked features, then a safe policy does not exist.*

So the robot can determine that no safe policy exists if it can *cover* $\mathcal{S}_{rel}$ using known-to-be-locked features.

Existence of safe policies. Relevant features of dominating policies ($\mathcal{S}_{rel}$) can help us determine non-existence of safe policies, but they do not directly help in finding a safe policy. I could focus on one safe policy that is most likely to be safe, similar to the simple most-likely-policy heuristic (Example 7.1). Instead, though, I 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. 7.1) 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, I can adapt to our needs tools in the literature for finding maximum feasible sets.

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

Definition 7.1. *A subset of unknown features $\Phi \subseteq \Phi_?^{\mathbf{R}}$ is an* **irreducible infeasible set (IIS)** *if 1) safe policies do not exist if Φ are locked features and $\Phi_?^{\mathbf{R}} \setminus \Phi$ are free features; 2) once any feature $\phi \in \Phi$ is known to be free (that is, $\Phi \setminus \{\phi\}$ are locked features and $\Phi_?^{\mathbf{R}} \setminus \Phi \cup \{\phi\}$ are free features), the robot can find a safe policy.*

I 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 $\mathcal{S}_{IIS}$ to be the set of IISs induced by all relevant unknown features. Formally,

$$\top \iff \forall \Phi \in \mathcal{S}_{IIS}, \Phi_F^{\mathbf{R}} \cap \Phi \neq \emptyset. \tag{7.6}$$

Example 7.3. *In Figure 7.2(b), $\mathcal{S}_{IIS} = \{\{\phi_1\}, \{\phi_2, \phi_3\}\}$. If the robot knows that ϕ_1 is a free feature, and at least one of ϕ_2 or ϕ_3 is a free feature, then a safe policy must exist.*

To compute $\mathcal{S}_{IIS}$ and $\mathcal{S}_{rel}$, I use Algorithm *DomPolicies* (Algorithm 6.1) to first find dominating policies and their relevant features, $\mathcal{S}_{rel}$.[1] Then to compute $\mathcal{S}_{IIS}$, I observe that in our problem,

$$\mathcal{S}_{IIS} = \{\Phi \subseteq \Phi_{rel} : |\Phi \cap \Phi'| = 1, \forall \Phi' \in \mathcal{S}_{rel}\}. \tag{7.7}$$

By the definition of IIS, when any Φ (defined in the equation above) are locked and $\Phi_?^{\mathbf{R}} \setminus \Phi$ are free, no safe policy exists because every dominating policy has exactly one relevant feature that is locked. However, hypothetically, if any $\phi \in \Phi$ is free, we know a safe policy exists because for some dominating policy, its only locked relevant feature is now free. Our implementation computes $\mathcal{S}_{rel}$ first and then uses the equation above to compute $\mathcal{S}_{IIS}$.

7.2.3 Set-Cover-Based Algorithm

Our set-cover-based algorithm specializes Algorithm 7.1 by using query responses to not only update the partition of features, but to also update $\mathcal{S}_{IIS}$ and $\mathcal{S}_{rel}$. Denote the two sets under partition ψ by $\mathcal{S}_{IIS}^{\psi}, \mathcal{S}_{rel}^{\psi}$. Specifically, it initially computes $\mathcal{S}_{IIS}^{\psi_0}$ and $\mathcal{S}_{rel}^{\psi_0}$ under the initial partition ψ_0. It needs to maintain these two sets in the query process. Suppose the robot has partition ψ and queries about feature ϕ_q. If ϕ_q is free, then the sets in $\mathcal{S}_{IIS}^{\psi}$ that contain ϕ_q are covered, so they are removed from $\mathcal{S}_{IIS}^{\psi}$. Also, since ϕ_q is free, it cannot be used to cover sets in $\mathcal{S}_{rel}^{\psi}$. I remove it from the sets in $\mathcal{S}_{rel}^{\psi}$, meaning that the sets originally containing ϕ_q need to be covered by some other feature. Formally,

$$\mathcal{S}_{IIS}^{\psi'} \leftarrow \mathcal{S}_{IIS}^{\psi} \setminus \{\Phi \in \mathcal{S}_{IIS}^{\psi} : \phi_q \in \Phi\}; \tag{7.8}$$

$$\mathcal{S}_{rel}^{\psi'} \leftarrow \{\Phi \setminus \{\phi_q\} : \Phi \in \mathcal{S}_{rel}^{\psi}\}. \tag{7.9}$$

[1] Note that this is different from Φ_{rel} defined in Ch. 6. Φ_{rel} is the union of the relevant features of all dominating policies, that is, $\Phi_{rel} = \cup \mathcal{S}_{rel}$. For example, in Figure 7.2 (a), $\mathcal{S}_{rel} = \{\{\phi_1, \phi_2\}, \{\phi_1, \phi_3\}\}$; $\Phi_{rel} = \{\phi_1, \phi_2, \phi_3\}$.

Similarly, if ϕ_q is locked, I update both sets as follows.

$$\mathcal{S}^{\psi'}_{rel} \leftarrow \mathcal{S}^{\psi}_{rel} \setminus \{\Phi \in \mathcal{S}^{\psi}_{rel} : \phi_q \in \Phi\}; \tag{7.10}$$

$$\mathcal{S}^{\psi'}_{IIS} \leftarrow \{\Phi \setminus \{\phi_q\} : \Phi \in \mathcal{S}^{\psi}_{IIS}\}. \tag{7.11}$$

Finally, for the set-cover-based version of Algorithm 7.1, in Line 13, I can avoid the expense of solving Eq. 7.1, because $\top$ (or $\bot$) is true simply if $\mathcal{S}^{\psi}_{IIS}$ (or $\mathcal{S}^{\psi}_{rel}$) has become empty, meaning it is fully covered. The algorithm is described in Algorithm 7.2.

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

Adaptive submodularity. While I have shown $\top$ and $\bot$ are each equivalent to a set cover problem, I 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 7.2, it cannot assert ϕ_1 is a locked feature so that it can cover $\mathcal{S}_{IIS}$ 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 ψ:

$$f_{IIS}(\psi) = |\{\Phi \in \mathcal{S}^{\psi_0}_{IIS} : \Phi^{\psi}_F \cap \Phi \neq \emptyset\}| \tag{7.12}$$

$$f_{rel}(\psi) = |\{\Phi \in \mathcal{S}^{\psi_0}_{rel} : \Phi^{\psi}_L \cap \Phi \neq \emptyset\}| \tag{7.13}$$

I further define the one-step gain in coverage as follows, similar to Δ of Golovin and Krause (2011). $\Delta_{IIS}(\phi|\psi) := p_F(\phi) \cdot |\mathcal{S}^{\psi}_{IIS}[\phi]|; \Delta_{rel}(\phi|\psi) := p_L(\phi) \cdot |\mathcal{S}^{\psi}_{rel}[\phi]|$, where $\mathcal{S}^{\psi}_{IIS}[\phi]$ is the number of sets in $\mathcal{S}^{\psi}_{IIS}$ that contain ϕ; $\mathcal{S}^{\psi}_{rel}[\phi]$ is defined similarly. I now show that f_{IIS} and f_{rel} are *adaptive submodular functions* (Golovin and Krause 2011) based on the subset relation: $\psi \subseteq \psi' \iff \Phi^{\psi}_F \subseteq \Phi^{\psi'}_F \wedge \Phi^{\psi}_L \subseteq \Phi^{\psi'}_L$.

Theorem 7.1. *f_{IIS} and f_{rel} are both adaptive submodular functions under $\subseteq$.*

Proof. I prove that f_{IIS} is an adaptive submodular function. The proof for f_{rel} is nearly identical.

It is sufficient to show that for any ψ, ψ' where $\psi \subseteq \psi'$ and any feature ϕ, $\Delta_{IIS}(\phi|\psi) \geq \Delta_{IIS}(\phi|\psi')$. I observe that $\Delta_{IIS}(\phi|\psi) = p_F(\phi)|\mathcal{S}^{\psi}_{IIS}[\phi]|$ if $\phi \in \Phi^{\psi}_?$, and is 0 otherwise. Since $\psi \subseteq \psi'$, $\Phi^{\psi}_? \supseteq \Phi^{\psi'}_?$, that is, unknown features monotonically vanish over time. So the following are the three possible relations between $\phi, \Phi^{\psi}_?$ and $\Phi^{\psi'}_?$.
1) $\phi \in \Phi^{\psi}_?$ and $\phi \in \Phi^{\psi'}_?$: $\Delta_{IIS}(\phi|\psi) = p_f(\phi)|\mathcal{S}^{\psi}_{IIS}[\phi]| \geq \Delta_{IIS}(\phi|\psi') = p_f(\phi)|\mathcal{S}^{\psi'}_{IIS}[\phi]|$ since $\mathcal{S}^{\psi}_{IIS}[\phi] \supseteq \mathcal{S}^{\psi'}_{IIS}[\phi]$.

Algorithm 7.2 Set-Cover-Based Myopic Query Selection

given query selection criterion h, initial partitions $\Phi_F^{\psi_0}, \Phi_L^{\psi_0}, \Phi_?^{\psi_0}$
$\psi \leftarrow \psi_0$
while not ($\mathcal{S}_{IIS} = \emptyset$ or $\mathcal{S}_{rel} = \emptyset$) **do**
    choose ϕ_q according to h based on partition ψ
    query the human about ϕ_q
    $\Phi_?^{\psi'} \leftarrow \Phi_?^{\psi} \setminus \{\phi_q\}$
    if ϕ_q is free **then**
        $\Phi_F^{\psi'} \leftarrow \Phi_F^{\psi} \cup \{\phi_q\}$
        $\mathcal{S}_{IIS}^{\psi'} \leftarrow \mathcal{S}_{IIS}^{\psi} \setminus \{\Phi \in \mathcal{S}_{IIS}^{\psi} : \phi_q \in \Phi\}$
        $\mathcal{S}_{rel}^{\psi'} \leftarrow \{\Phi \setminus \{\phi_q\} : \Phi \in \mathcal{S}_{rel}^{\psi}\}$
    else ▷ ϕ_q is locked
        $\Phi_L^{\psi'} \leftarrow \Phi_L^{\psi} \cup \{\phi_q\}$
        $\mathcal{S}_{rel}^{\psi'} \leftarrow \mathcal{S}_{rel}^{\psi} \setminus \{\Phi \in \mathcal{S}_{rel}^{\psi} : \phi_q \in \Phi\}$
        $\mathcal{S}_{IIS}^{\psi'} \leftarrow \{\Phi \setminus \{\phi_q\} : \Phi \in \mathcal{S}_{IIS}^{\psi}\}$
    end if
    $\psi \leftarrow \psi'$
    update $\top$ or $\bot$ based on the new partition
end while
if $\mathcal{S}_{IIS} = \emptyset$ **then** ▷ $\top$ is true
    return safely-optimal policy under ψ by solving Eq. 7.1
else ▷ $\bot$ is true
    return "No safe policy exists."
end if

2) $\phi \notin \Phi_?^{\psi}$ and $\phi \notin \Phi_?^{\psi'}$: $\Delta_{IIS}(\phi|\psi) = \Delta_{IIS}(\phi|\psi') = 0$.
3) $\phi \in \Phi_?^{\psi}$ and $\phi \notin \Phi_?^{\psi'}$: $\Delta_{IIS}(\phi|\psi) = p_f(\phi)|\mathcal{S}_{IIS}^{\psi}[\phi]| \geq \Delta_{IIS}(\phi|\psi') = 0$.

So in all cases, $\Delta_{IIS}(\phi|\psi) \geq \Delta_{IIS}(\phi|\psi')$, which completes the proof. □

One benefit of this result is the following. Golovin and Krause (2011) implied that if the robot knows that it can cover $\mathcal{S}_{IIS}$, it is approximately-optimal to choose $\arg\max_\phi \Delta_{IIS}(\phi|\psi)$. Similarly, if it knows that it can cover $\mathcal{S}_{rel}$, the robot should choose $\arg\max_\phi \Delta_{rel}(\phi|\psi)$. However, in our problem, the robot does not know which outcome ($\top$ or $\bot$) is true before it finishes querying. So I 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-coverage-ratio) heuristic is more expensive to compute. I will empirically test to see if the second one finds a better query that is worth the extra computation in Sec. 7.3.

Set-cover heuristic (h_{SC}). One simple way of combining two objectives is the sum of both, weighted by the number of sets remains to cover.

$$\phi_q = \arg\max_{\phi} h_{SC}(\phi; \psi) \tag{7.14}$$

$$h_{SC}(\phi; \psi) = \frac{\Delta_{IIS}(\phi|\psi)}{|\mathcal{S}^{\psi}_{IIS}|} + \frac{\Delta_{rel}(\phi|\psi)}{|\mathcal{S}^{\psi}_{rel}|} \tag{7.15}$$

I 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 $\mathcal{S}_{IIS}$ (or $\mathcal{S}_{rel}$) by maximizing Δ_{IIS} (or Δ_{rel}), which Golovin and Krause (2011) show to be approximately-optimal. When the probabilities of changeability vary, I no longer have a pure set cover problem and it is difficult to provide a theoretical guarantee. The robot's strategy is to query about the feature that makes the most progress (in expectation) on *both* set cover problems at once. I see in the following example that h_{SC} does find an optimal query policy in the example in Figure 7.2.

Example 7.4. *I consider again the domain in Figure 7.2.* $p_F(\phi_1) = 0.5$, $p_F(\phi_2) = p_F(\phi_3) = 0.6$. *Consider the first query to pose,* $h_{SC}(\phi_1; \psi_0) = 0.5 \cdot 1/2 + 0.5 \cdot 2/2 = 0.75$. $h_{SC}(\phi_2; \psi_0) = h_{SC}(\phi_3; \psi_0) = 0.6 \cdot 1/2 + 0.4 \cdot 1/2 = 0.5$. *Higher heuristic values mean more coverage. So the algorithm would query about* ϕ_1. *I can verify that* h_{SC} *finds the optimal query policy (the same as Figure 7.1 (c)).*

Inverse-coverage-ratio heuristic (h_{ICR}). The method I describe below uses some properties of f_{IIS} and f_{rel} (Eqs. 7.12 and 7.13). I first describe a theorem adapted from Golovin and Krause (2011). Let $c(\pi^*_q|\psi)$ be the expected number of features queried by an optimal query policy π^*_q starting from ψ.

Theorem 7.2. *(Adapted from Lemma A.9 in Golovin and Krause (2011)) Starting from partition* ψ, *the optimal query policy* π^*_q *has the following property,*

$$c(\pi^*_q|\psi) \geq \frac{\Delta_{IIS}(\pi^*_q|\psi)}{\max_{\phi} \Delta_{IIS}(\phi|\psi)}. \tag{7.16}$$

where $\Delta_{IIS}(\pi^*_q|\psi)$ is the number of IISs π^*_q can cover in expectation. Clearly, $\Delta_{IIS}(\pi^*_q|\psi) \geq \mathbb{P}[\top; \psi] \cdot |\mathcal{S}^{\psi}_{IIS}|$. $\mathbb{P}[\top; \psi]$ is the probability that safe policies exist starting from partition ψ. This is to say, with probability $\mathbb{P}[\top; \psi]$, π^*_q needs to cover all sets in

$\mathcal{S}_{IIS}$ (because a safe policy indeed exists). Combining both, I have

$$ICR_{IIS}(\psi) := \frac{\mathbb{P}[\top;\psi]\cdot|\mathcal{S}^{\psi}_{IIS}|}{\max_{\phi}\Delta_{IIS}(\phi|\psi)} \leq c(\pi^{*}_{q}|\psi) \tag{7.17}$$

I call the left-hand-side the *inverse coverage ratio* for $\mathcal{S}_{IIS}$. Intuitively, the inequality says the following. Using one query, I can greedily-optimally cover $\max_{\phi}\Delta_{IIS}(\phi|\psi)$ sets in $\mathcal{S}_{IIS}$ in expectation. Optimistically, I may cover the same number of sets in the following queries as well, but not more than this number because of adaptive submodularity. So to cover all the sets that it will cover, at least $\Delta_{IIS}(\pi^{*}_{q}|\psi)/\max_{\phi}\Delta_{IIS}(\phi|\psi)$ queries need to be asked in expectation. This is an optimistic estimation, which serves as a lower bound for $c(\pi^{*}_{q}|\psi)$.

I define ICR_{rel} similarly, which has a similar property.

$$ICR_{rel}(\psi) := \frac{\mathbb{P}[\bot;\psi]\cdot|\mathcal{S}^{\psi}_{rel}|}{\max_{\phi}\Delta_{rel}(\phi|\psi)} \leq c(\pi^{*}_{q}|\psi) \tag{7.18}$$

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

$$ICR(\psi) = ICR_{IIS}(\psi) + ICR_{rel}(\psi) \tag{7.19}$$

Note that ICR is not necessarily a lower bound, but intuitively it serves as an estimate of how many queries need to be asked starting from ψ, considering how many queries are needed when safe policies exist (ICR_{IIS}) and when safe policies do not exist (ICR_{rel}). The query selection criterion is hence the following.

$$\phi_{q} = \arg\min_{\phi} h_{ICR}(\phi;\psi) \tag{7.20}$$

$$\begin{aligned} h_{ICR}(\phi;\psi) =& p_{F}(\phi)ICR(\psi|\phi\in\Phi^{\mathbf{R}}_{F}) \\ &+ p_{L}(\phi)ICR(\psi|\phi\in\Phi^{\mathbf{R}}_{L}) \end{aligned} \tag{7.21}$$

where $\psi|\phi\in\Phi^{\mathbf{R}}_{F}$ is the partition of features that, starting from partition ψ, the robot queries about ϕ and confirms that it is a free feature. $\psi|\phi\in\Phi^{\mathbf{R}}_{L}$ is defined similarly. It considers the probability that a feature ϕ is free or locked, and uses ICR to estimate the number of queries needed under the partition where ϕ is free or locked. This is more expensive to compute compared with h_{SC} since I need to compute $\mathbb{P}[\top;\psi]$ and $\mathbb{P}[\bot;\psi]$. I will empirically test if h_{ICR} finds better queries than h_{SC} so it is worth the computation. In the following example, h_{ICR} finds the optimal query while h_{SC} does not.

Example 7.5. *In Figure 7.2, let* $p_{F}(\phi_{1}) = 0.9$, $p_{F}(\phi_{2}) = p_{F}(\phi_{3}) = 0.1$. $h_{SC}(\phi_{1}) =$

$.9 * 1/2 + .1 * 2/2 = .55$*;* $h_{SC}(\phi_2) = h_{SC}(\phi_3) = .1 * 1/2 + .9 * 1/2 = .5$*. So* h_{SC} *would first query about* ϕ_1*. However, since safe policies are unlikely to exist, by computing* $\mathbb{P}[\top]$ *and* $\mathbb{P}[\bot]$*,* h_{ICR} *would query about* ϕ_2 *and (if it's locked) then* ϕ_3 *aiming to prove that no safe policy exists, which asks fewer queries in expectation.*

In detail, h_{SC} *asks about* ϕ_1 *first, and then about* ϕ_2 *when* ϕ_1 *is free (with probability* 0.9*), and then about* ϕ_3 *when* ϕ_2 *is locked (with probability* $0.9 * 0.9$*). So it asks about* $1 + 0.9 * 1 + 0.9 * 0.9 * 1 = 2.71$ *features in expectation.*

h_{ICR} *asks about* ϕ_2 *first. Depending on the response, it will ask about* ϕ_1 *(if* ϕ_2 *is free) or* ϕ_3 *(if* ϕ_2 *is locked). When* ϕ_2 *is locked and* ϕ_3 *is free (this happens with probability* $0.9 * 0.1$*), it will ask about* ϕ_1*. So* h_{ICR} *asks about* $1 + 1 + 0.9 * 0.1 * 1 = 2.09$ *features in expectation.*

7.3 Empirical Evaluation

In this section, I want to empirically answer the following three questions with uniformly-randomly generated unknown features.

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?

I consider a variation of Ch. 6's robot navigation domain for evaluation, illustrated in Figure 7.3 (left). The size of the domain is 6×6, with 5 randomly-generated walls which the robot cannot move through so I have various different dynamics. The robot starts from the bottom-left corner and is asked to turn off a switch in the top-right corner. It can move in all cardinal directions to its adjacent cell in a time step, unless blocked by a wall or border of the domain. The reward is 1 if the robot turns off the switch and 0 in all other states. $\gamma = 0.9$.

The robot is required to turn off the switch. This is implemented by setting S_G and δ_G. In general, it may be difficult to enforce a guarantee on the robot's performance using S_G and δ_G. It is however possible in this simple domain: I set $\delta_G > 0$ where S_G are the states where the switch is off. We can see that as long as $\delta_G > 0$, the robot will reach S_G with some probability. As the only positive reward comes from turning off the switch, and the robot needs to find a safe policy to reach the switch (since $\delta_G \geq 0$), the safely-optimal policy would deterministically turn off the switch. Here I set $\delta_G = 0.1$.

There are carpets randomly placed in this environment. 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.

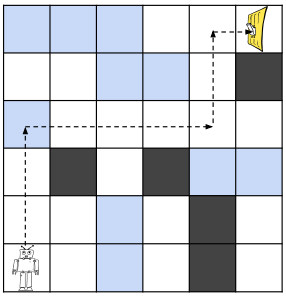


Figure 7.3: The robot navigation domain. The blue-colored tiles are carpets and dark tiles are walls. The dashed-line policy is safe if the robot knows that the traversed carpets are free to change.

I compare several (heuristic) strategies for query selection. One, **Optimal**, finds the exact optimal query policy. It evaluates $c(\pi_q^*|\psi)$, the minimum number of queries needed in expectation starting from ψ, for all ψ that are reachable by any query policy. This can be done using dynamic programming by observing the following fact: Under a partition ψ where $\top$ or $\bot$ is not true, $c(\pi_q^*|\psi) = \min_\phi p_F(\phi)c(\pi_q^*|(\psi|\phi \in \Phi_F^{\mathbf{R}})) + p_L(\phi)c(\pi_q^*|(\psi|\phi \in \Phi_L^{\mathbf{R}})) + 1$.

I also compare to variations of h_{SC} that aim to cover $\mathcal{S}_{IIS}$ and $\mathcal{S}_{rel}$, respectively: h_{SC} **(IIS)** queries about

$$\arg\max_{\phi \in \Phi_?^{\mathbf{R}}} \Delta_{IIS}(\phi|\psi)/|\mathcal{S}_{IIS}^{\psi}|. \tag{7.22}$$

h_{SC} **(rel)** queries about

$$\arg\max_{\phi \in \Phi_?^{\mathbf{R}}} \Delta_{rel}(\phi|\psi)/|\mathcal{S}_{rel}^{\psi}|. \tag{7.23}$$

I have also described the **most-likely-policy** in the simple heuristics section. Finally, I also compare against the following heuristics that are based on the probability of existence of safe policies. **probability-maximization ($\top$)** finds the feature that, if known be to free, *increases* the probability of existence of safe policies the most:

$$\arg\max_{\phi \in \Phi_?^{\mathbf{R}}} p_F(\phi)\mathbb{P}[\top; (\psi|\phi \in \Phi_F^{\mathbf{R}})]. \tag{7.24}$$

Similarly, **probability-maximization ($\bot$)** finds the feature that, if known to be locked,

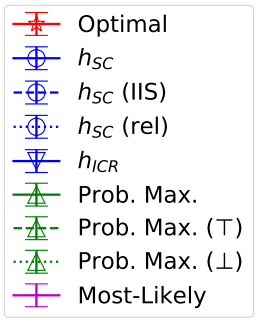


Figure 7.4: Legend for the following figures.

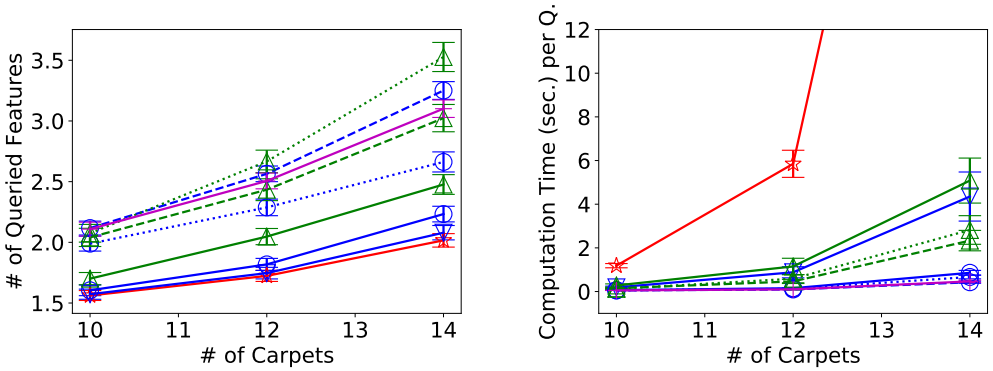


Figure 7.5: (Left) The number of queried features vs. the number of unknown features (carpets). (Right) Computation time per query vs. the number of carpets.

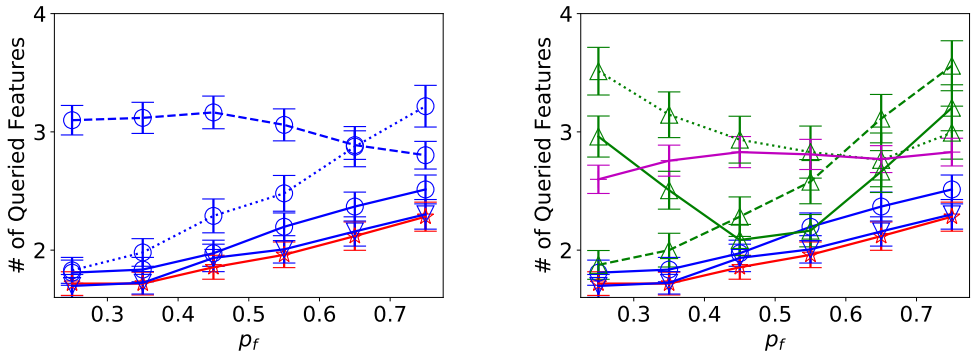


Figure 7.6: The number of queried features vs. p_F. The horizontal axis is the midpoint of the intervals where p_F is sampled from, namely, $[0, 0.5], [0.1, 0.6], \ldots$. To see the differences clearly, I report a subset of algorithms in each figure.

decreases the probability of existence of safe policies the most:

$$\arg\max_{\phi \in \Phi^{\mathbf{R}}_{?}} p_L(\phi)\mathbb{P}[\bot; (\psi | \phi \in \Phi^{\mathbf{R}}_{L})]. \tag{7.25}$$

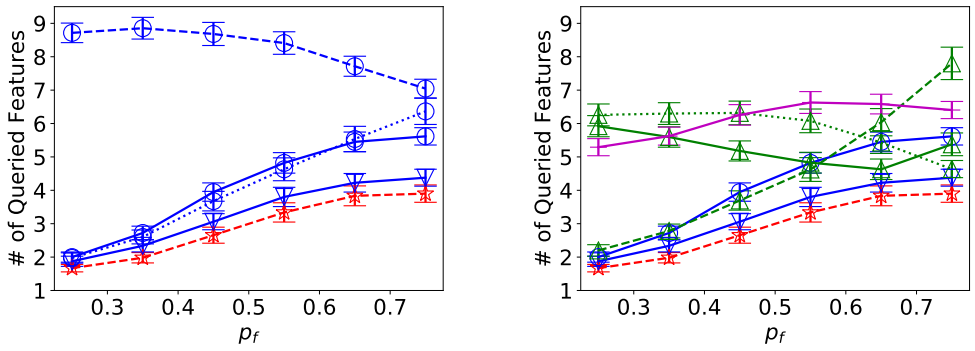


Figure 7.7: Randomly placed 40 carpets and 20 walls in a 10×10 domain. Optimal is intractable to run and is not shown.

Lastly, **probability-maximization** optimizes the joint objective of the previous two.

$$\arg\max_{\phi \in \Phi_?^{\mathbf{R}}} p_F(\phi)\mathbb{P}[\top; (\psi|\phi \in \Phi_F^{\mathbf{R}})] + p_L(\phi)\mathbb{P}[\bot; (\psi|\phi \in \Phi_L^{\mathbf{R}})]. \tag{7.26}$$

I evaluate these selection strategies in the following experiments. In the first experiment, I report the performance of all strategies while varying the numbers of unknown features (numbers of carpets in this example). In the second experiment, I fix the number of carpets and show the performance of all strategies under different distributions of p_F.

Experiment 1: Varying the number of carpets. I first consider a uniformly random p_F setting with a varying number of carpets. The values of $p_F(\phi), \phi \in \Phi_?^{\mathbf{R}}$ are uniformly randomly generated between $[0, 1]$. The numbers of carpets are 10, 12, 14. I 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 7.5.

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 the performance closest to optimal in the left figure. By using more computation to estimate the number of features needed to cover each set in expectation, h_{ICR} is closer to the optimal than h_{SC}. We see in the right figure that 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 methods, in the left figure, including variations of h_{SC} that focus on one set-cover problem, probability-maximization and its variations, and most-likely-policy (answering Q2). I find h_{ICR} has performance closer to optimal compared with h_{SC} while using only slightly more computation time in the right figure (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, I consider generating p_F in small intervals (Figure 7.6). The size of intervals are 0.5, that is, the intervals are $[0, 0.5], [0.1, 0.6]$, and so on.

h_{ICR} and h_{SC} still have performance closest to optimal (answering Q1). They are also robust to different distributions of p_F. Unsurprisingly, h_{SC} (IIS) performs well when p_F values are large (so that safe policies are indeed likely to exist), and h_{SC} (rel) performs well when p_F values are small. Thus, the answer to Q2 is more nuanced: cheaper heuristics are competitive under particular narrower conditions, namely, when p_f is close to 0 (where h_{SC} (rel) and probability-maximization ($\top$) have close-to-optimal performances) and when p_f is close to 1 (where h_{SC} (IIS) and probability-maximization ($\bot$) have close-to-optimal performances). These cheaper heuristics target at either finding a safe policy or proving that no safe policy exists, while they do not consider both objectives. We also see that although probability-maximization has performance close to h_{SC} in Experiment 1, it actually has much worse performance under certain p_F ranges.

Experiment 3: Using approximated $\mathcal{S}_{IIS}, \mathcal{S}_{rel}$. It can be expensive to find all dominating policies and IISs (a similar observation made in Regan and Boutilier (2010)). So our algorithms would be slow to run in large domains. In this experiment, I use approximated $\mathcal{S}_{IIS}, \mathcal{S}_{rel}$ by early stopping when I compute dominating policies and IISs, and I recompute both sets after each response from the human. Since the robot only needs to decide what is the next feature to query about, it may not be necessary to compute the exact $\mathcal{S}_{IIS}$ and $\mathcal{S}_{rel}$. It computes $\mathcal{S}_{IIS}, \mathcal{S}_{rel}$ for a limited time and stops, and then selects queries based on possibly incomplete sets. To determine if safe policies are found or no safe policies exist, it cannot simply check if $\mathcal{S}_{IIS}$ or $\mathcal{S}_{rel}$ are empty as in Algorithm *SetCoverQuery* (since both sets can be incomplete), but solves the LP problem in Eq. 7.1. So it is guaranteed to terminate when a safe policy is found or no safe policies exist. I run our algorithms in a 10×10 domain with 30 randomly placed carpets. I only compute dominating policies and IISs for 5 seconds. In Figure 7.7, the result is very similar to Experiment 1 regardless of the size of the domain. h_{SC} and h_{ICR} still find queries closest to the optimal ones compared with other heuristics.

To summarize, I find both h_{SC} and h_{ICR} are robustly close to the optimal query when I 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, I find some other candidate methods also have good performance (for example, using h_{SC} (IIS) when

p_F values are large). These are also simpler cases since I can focus on covering either $\mathcal{S}_{IIS}$ or $\mathcal{S}_{rel}$.

7.4 Conclusion

In this chapter, I considered the problem of querying under safety-constraint uncertainty without knowing an initial safe policy. The objective is to use the minimum number of queries in expectation to either find a safe policy, or prove none exists. I cast this problem into a pair of set-cover problems and devised heuristics for a myopic-greedy approach that empirically perform nearly optimally without excessive computational costs, and outperform other candidate heuristics. This chapter complements the query selection algorithm in Ch. 6 (Algorithm 6.3, *MMRQ-k*), which assumes that there exists an initially-known safe policy.

This query selection algorithm may not be efficient in larger domains, which is the same issue as in in the previous chapter. One possible way to resolve this problem is that we can stop early as we are computing $\mathcal{S}_{IIS}$ and $\mathcal{S}_{rel}$ (Regan and Boutilier 2011b) and our algorithms still have good performance using incomplete sets. This evaluation is left for future work. It is also of interest to derive a performance bound for our heuristic h_{SC} and h_{ICR} similar to Golovin and Krause (2011). The optimality guarantee in Golovin and Krause (2011) cannot be applied directly since my algorithms aim to solve two set cover problems simultaneously. Lastly, the query selection objective in this chapter does not take the values of policies into consideration. It aims to find a safe policy using efficient querying, when one exists, without considering the value of the policy. A future direction is to incorporate values of policies, so that the robot should bias towards finding safe policies with higher values. This may be done using the *weighted* set cover formulation (Williamson and Shmoys 2011), where an unknown feature is weighted by the value of the safely-optimal policy if such a feature is known-to-be-free.

CHAPTER 8

Query Selection Under Joint Uncertainty

I have considered the settings where the robot plans and queries under only reward uncertainty (Ch. 5) and only uncertainty on changeability of features (Ch. 6, 7). I provided algorithms that find approximately-optimal or empirically good queries in these settings. In this chapter, I consider the setting where the robot is uncertain about both the true reward function and the changeability of features. Since our algorithms in the previous three chapters only consider one type of uncertainty and assume that the other type of uncertainty is absent, I cannot directly use the previous algorithms. Here I provide algorithms that pose queries and take both types of uncertainty into consideration.

8.1 Problem Statement

Let r^* be the true reward function. The robot initially has a set of possible reward functions, $\mathcal{R}$, and a prior belief on which is the true reward function r^*. I assume $r^* \in \mathcal{R}$, consistent with Ch. 3.1. The robot also has a partition of features as defined in Ch. 3.2, and prior probability of changeability of features (Ch. 7). After all querying has ended, the robot's policy is evaluated by the true reward function r^*. It also needs to guarantee that its behavior will not change any locked features.

I have used batch queries (like k-policy queries in Ch. 5 and k-feature queries in Ch. 6) and sequential queries (like the sequential feature queries in Ch. 7) in the previous chapters. In this chapter, the robot can expect to find better reward queries after posing some feature queries to reduce feature uncertainty, or *vice versa*. So I allow the robot to query sequentially. Concretely, the robot can either pose a reward query or a feature query at a time. It receives the human's response to the query it posed before it poses the next query, or decides to stop querying and follows the safely-optimal policy under the current belief.

- A reward query: The robot poses $R \subseteq \mathcal{R}$ of the robot's choice. The human provides

a binary response: $r^* \in R$ or $r^* \notin R$. I denote the space of possible reward queries by Q_R.

- A feature query: The robot poses one unknown feature, $\phi \in \Phi_?^{\mathbf{R}}$. The human responds with the category of this feature: $\phi \in \Phi_F^{\mathbf{R}}$ or $\phi \in \Phi_L^{\mathbf{R}}$ (the same as the feature queries in Ch. 7). I denote the space of possible reward queries by Q_Φ.

Note that the two types of queries described above elicit information about the true reward function and the true partition of features, respectively. For simplicity, I do not consider query languages that can simultaneously elicit both kinds of information. We can easily see that Q_R can only change the robot's belief on the true reward function but not partitions of features. Q_Φ can only change the robot's belief on partitions of features but not on the true reward function.

Note that I do not use queries like trajectory queries or policy queries in this chapter. When the human says she prefers one trajectory over another, it is unclear whether the trajectory she likes has higher value or the other trajectories are unsafe. This can correlate two types of uncertainty and complicate the problem, which will be left for future work.

Now I define the query selection problem formally. Let c_q be the cost of posing one query, which is known by the robot. I assume that the cost of posing any query is the same and there exists an initial safe policy. In the domain that we will consider in this chapter, the robot can simply stay in place, which is a trivial safe policy. I define a *query policy* in a similar way as in Ch. 7, that is, it selects one element in $Q_R \cup Q_\Phi$, or decides to terminate querying, given a reward belief and a partition of features. Suppose the robot follows a query policy q. The robot's objective is to find the query policy that optimizes the following objective.

$$\max_q \mathbb{E}_{R\sim\mathcal{R},\Phi\sim\Phi_?^{\mathbf{R}};q}\big[\max_{\pi\in\Pi_\Phi} V_R^\pi\big] - c_q\mathbb{E}|q|, \tag{8.1}$$

I use the notation $R \sim \mathcal{R}, \Phi \sim \Phi_?^{\mathbf{R}}; q$ to denote the event that, after posing queries by following querying policy q, R are the reward functions that are consistent with the human's response and Φ are known-to-be-free features. The objective is the expected value of the posterior safely-optimal policy after querying minus the cost of querying (which is c_q times the expected number of posed queries).

8.2 Query Selection Methods

Since I consider sequential query policies, the number of query policies can be exponential in the number of possible queries. So it is impractical to enumerate all possible

query policies to find the optimal one. Similar to the query selection methods in Ch. 7, I consider heuristics that decide what is the next query to pose based on the current belief on the reward function and partition of features.

8.2.1 Myopic Heuristic

Since I have solutions to the problems of finding reward queries and finding feature queries respectively, I first consider a heuristic that finds myopically-optimal queries for both types, then poses the one that has the higher value. However, previous chapters (Ch. 5, 6 and 7) assumes that the robot selects queries under one type of uncertainty. So I need to modify the algorithms so that they can handle presence of both types of uncertainty. The main idea is that the robot computes a myopic query of one type without considering future queries that can possibly reduce the other type of uncertainty.

I first consider how to select a reward query. I use the greedy construction algorithm for policy queries Algorithm 5.1 to compute an approximately-optimal policy query and project it into the space of reward queries. To handle feature uncertainty, the robot selects a reward query without considering future queries that reduce feature uncertainty. It simply encodes the changeability of features into the transition function.

Next, I consider how to select a feature query. I adapt the query selection heuristic h_{SC} (Eq. 7.15) since it finds sequential queries, although its objective is different. The algorithm selects feature queries to find an initial safe policy. I modify the objective to be minimizing the number of queries to find *additional* safe policies rather than finding an initial safe policy. To handle reward uncertainty, I assume the reward function is the mean reward function under the current reward belief.

To decide between posing a myopically-optimal reward query or a myopically-optimal feature query, the robot computes both of their EVOI values (Viappiani and Boutilier 2010) and poses the query with the higher EVOI value. It stops querying when the EVOI value of the better query is less than the querying cost (similar the query selection criterion in Cohn et al. (2010)). The algorithm is described in Algorithm 8.1.

However, such a straightforward heuristic has two drawbacks: First, it completely ignores how the belief could change by posing the other kind of query at all. It only selects feature queries under $\overline{R}$, and selects reward queries without considering future possible queries about the features. Second, it may terminate early whenever no myopic improvement is possible. For example, even if posing a feature query has no immediate improvement on the robot's policy, the robot could benefit from posing multiple feature queries.

Example 8.1. *In Figure 8.1, the robot receives a reward of 1 when reaching a switch. It*

Algorithm 8.1 Myopic heuristic

while True **do**
    $q_R \leftarrow$ reward query found by the query selection algorithm in Ch. 5 under the transition function that encodes the changeability of features
    $q_\Phi \leftarrow$ feature query found using the query selection algorithm in Ch. 7 under the mean reward function under the current reward belief
    if $EVOI(q_R) < c_q$ and $EVOI(q_\Phi) < c_q$ **then**
        break
    else if $EVOI(q_R) \geq EVOI(q_\Phi)$ **then**
        ask q_R
    else
        ask q_Φ
    end if
end while

is uncertain about which switch it should turn off. So there are two reward functions and the prior reward belief is uniform (there is a 50% chance that it gets the reward of 1 by turning off each switch). To reach the switch, it needs to traverse some carpets. Each carpet corresponds to an unknown feature. Each carpet has a probability of 0.5 of being free.

Using the myopic heuristic, the robot finds that it cannot find a better safe policy by posing a single reward query or feature query. So $EVOI$ *is* 0 *for all queries. The robot would not pose any query. This is clearly a suboptimal query policy if the cost of querying is small. When* c_q *is small, the optimal query policy is to first pose a reward query to identify which switch has the reward of 1, and then query about the carpets that the robot needs to traverse to reach the switch.*

Knowing which switch has the positive reward requires one reward query. After that, the robot may pose one or two feature queries depending on the human's response. So the objective value is $1/4 * 1 - c_q(1 + 1 + 1/2) = 1/4 - (5/2)c_q$*. This is the optimal query policy if* $c_q < 1/10$*.*

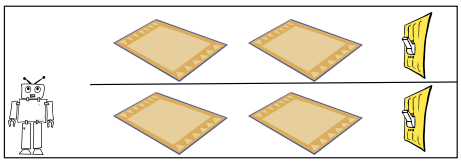


Figure 8.1: An example where the myopic heuristic finds a suboptimal query policy.

8.2.2 Dominating-Policy-Based Heuristic

Myopic queries may terminate early when there does not exist a single query that can possibly improve the robot's policy. I consider another extreme: the robot has a policy in mind and aims to prove that it is safely-optimal. Concretely, it finds a *dominating policy* (using Algorithm 6.1, *DomPolicies*), and queries if the reward functions it dominates contain the true reward function and if its relevant features are free. It repeats this process until it finds that the policy is not safely-optimal (because it is unsafe or the reward functions it dominates do not contain the true reward function), in which case it finds another dominating policy and repeats this process, or decides to stop querying.

I implement this idea using a modified definition of dominating policies. I define dominating policies in our problem to be the set of policies the robot may consider under both types of uncertainty. The robot needs to decide which dominating policy to querying about. So for a dominating policy π that optimizes $R \subseteq \text{support}(\psi)$ where Φ are its relevant features, I set its weight to be

$$w(\pi) = p_f(\pi)\mathbb{P}[R;\psi](V_R^{\pi} - V_R^{\pi_0} - c_q(|\Phi_{rel}(\pi)| + 1_{[0<|R|<|support(\psi)|]})), \tag{8.2}$$

where π_0 is the safely-optimal policy without further querying; $1_{[x]}$ is an indicator function, which is 1 when x is true and 0 otherwise. The weight is the product of the probability that π is a safe policy, the probability that R contains the true reward function, and the improvement of the policy value minus the querying cost. This is formally described in Algorithm 8.2.

Algorithm 8.2 Dominating-policy-based heuristic

$\Gamma \leftarrow$ all dominating policies (computed using Algorithm *DomPolicies* in Ch. 6) under all possible reward beliefs
while $\Gamma \neq \emptyset$ **do**
    compute the weights, w, of all dominating policies using Eq. 8.2
    $\pi \leftarrow \max_{\pi'} w(\pi')$
    if $w(\pi) > 0$ **then**
        $q_R \leftarrow$ the set of reward functions dominated by π
        $q_\Phi \leftarrow \Phi_{rel}(\pi)$
        pose the queries in $q_R \cup q_\Phi$, prioritized by their EVOI values
    else
        break
    end if
    recompute Γ under the updated belief
end while

First, I observe that this heuristic finds a better query than the myopic heuristic in the example in Figure 8.1.

Example 8.2. *Consider the example in Figure 8.1. Using the dominating-policy-based heuristic, there are three dominating policies. Two of them have the same weights:* $1/4 * 1/2 * (1 - 0 - c_q * 3)$. *The other dominating policy (that does not reach any switches) has the weight of 0. So the robot would focus on querying about one of the two dominating policies that have positive weights. If the dominating policy is unsafe or the switch that it reaches has 0 return, the robot would focus on querying about the other dominating policy.*

However, the dominating-policy-based heuristic is not guaranteed to find the optimal query policy.

Example 8.3. *The dominating-policy-based heuristic can find suboptimal queries when it focuses on the policy that is most promising to be safely-optimal (computed using Eq 8.2). In Figure 8.2, suppose the probabilities of changeability of carpets are the same, and the reward belief is uniform. The robot would focus on* π_2 *because it achieves the largest reward in expectation: its weight is* $1/2 * 1 * (0.6 - 0 - c_q * 1) = 0.3 - 1/2 * c_q$. *The weights of* π_1 *and* π_3 *are smaller, which are* $1/2 * 1/2 * (1 - 0 - c_q * 2) = 0.25 - 1/2 * c_q < 0.3 - 1/2 * c_q$.

However, if the querying cost is small, the query policy would only query about π_1 *and* π_3. *The robot poses a reward query to decide* π_1 *or* π_3 *has a higher value, and then queries about the carpet it needs to traverse.*

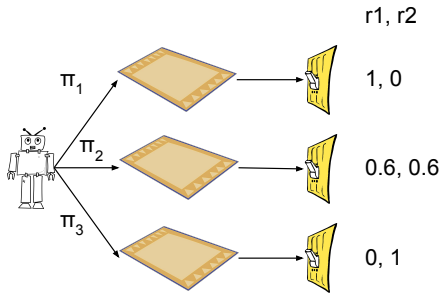


Figure 8.2: An example where the dominating-policy-based heuristic finds a suboptimal query policy.

8.2.3 Batch-Query-Based Heuristic

From the example in Figure 8.2, we can see that the dominating-policy-based heuristic fails to find an optimal query policy because it only focuses on querying about one dominating policy, which may not be the best policy to query about. The robot may pose a

better query when it considers multiple dominating policies to decide which query to pose. I exploit this idea and design a batch-query-based heuristic in this section.

In the previous work, I used a greedy construction approach to find a set of useful policies (Algorithm 5.1). I generalize that method to this setting. The difference here is that I need to consider how many relevant features a policy has and the cost of posing feature queries to verify if this policy is safe. I adapt the definition of $EVOI$ here:

$$EVOI(q_\Pi) = \mathbb{E}_{r\sim\mathcal{R}}\mathbb{E}_{\Phi\sim\Phi^{\mathbf{R}}_?} \max_{\pi\in q_\Pi:\Phi_{rel}(\pi)\subseteq\Phi} V^\pi_r - c_q|\cup_{\pi\in q_\Pi}\Phi_{rel}(\pi)|. \tag{8.3}$$

The right-hand-side is the expected value of the safely-optimal policy after the response minus the cost of queries. One challenge of optimizing this objective is that I need to sample all partitions of unknown features ($\Phi \sim \Phi^{\mathbf{R}}_?$). So I approximate this objective in a similar way to what I did for myopic heuristics, that is, I encode the changeability of unknown features into the transition function:

$$\overline{EVOI}(q_\Pi) = \mathbb{E}_{r\sim\mathcal{R}} \max_{\pi\in q_\Pi} \tilde{V}^\pi_r - c_q|\cup_{\pi\in q_\Pi}\Phi_{rel}(\pi)|. \tag{8.4}$$

Here, $\tilde{V}^\pi_r$ is the value function that encodes the changeability of $\Phi^{\mathbf{R}}_?$ into the transition function.

To select one query, the robot finds a binary-policy query that optimizes $\overline{EVOI}$ following a procedure similar to the greedy construction method for policy queries (Algorithm 5.1). First, the robot finds the safely-optimal policy under the mean reward function while allowed to change some unknown features. When it changes an unknown feature, it includes the cost of querying in the objective. So the objective is

$$\max_\pi \mathbb{E}_{r\sim\mathcal{R}}\tilde{V}^\pi_r - c_q|\Phi_{rel}(\pi)|. \tag{8.5}$$

Let the solution to the problem above be π_0. Then, the robot solves the following optimization problem to find the second policy. It complements the previous policy by maximally improving the objective. Indeed, I could use an objective similar to optimizing EUS (Sec. 5.2.1):

$$\max_\pi \mathbb{E}_{r\sim\mathcal{R}} \max(\tilde{V}^\pi_r - \tilde{V}^{\pi_0}_r, 0) - c_q|\Phi_{rel}(\pi) \setminus \Phi_{rel}(\pi_0)|.$$

However, it is unfair to compare $\tilde{V}^\pi_r$ with $\tilde{V}^{\pi_0}_r$ if π does not change some features that are changed by π_0. So I take this into consideration. I find the second policy by solving the

following optimization problem:

$$\max_{\pi} \mathbb{E}_{r\sim\mathcal{R}} \max\left(\tilde{V}_r^{\pi} - 1_{[\Phi_{rel}(\pi_0)\subseteq\Phi_{rel}(\pi)]}\tilde{V}_r^{\pi_0}, 0\right) - c_q|\Phi_{rel}(\pi) \setminus \Phi_{rel}(\pi_0)|. \tag{8.6}$$

where $1[\cdot]$ is an indicator function. Compared with the MILP objective in Sec. 5.2.1, I only subtract the value of the previous policy, $\tilde{V}_r^{\pi_0}$, when it is a safe policy if π is a policy. These objectives are equivalent to MILP problems. The technical details are deferred to Sec. 8.4.

After finding q_{Π}, the robot can query about either the reward functions they dominate or their relevant features. In this heuristic, the robot selects the one that has the highest $EVOI$ value. Note that, different from the myopic heuristic, this heuristic may select a query with $EVOI$ of 0 (which means that the $EVOI$ values of all candidate queries are 0, in which case the tie is broken randomly). In contrast to the myopic heuristic, a query may have no immediate value, but the robot may still pose it and believe that it can benefit from asking more queries. Moreover, since the sequential querying process is adaptive, its query policy can have at least as much $EVOI$ as that of a single multiple-response-policy query in expectation.

In the following example, I show that the batch-query-based heuristic does find optimal queries in the previous two examples.

Example 8.4. *In Figure 8.1, the two policies would correspond to two dominating policies where each of them reaches a different switch. In the example in Figure 8.2, the batch-query-based heuristic identifies that π_1 and π_3 are worth querying about. Concretely, the first policy it adds is π_2. After that it would add π_1 (or π_3). Then it runs query iteration, which replaces π_2 with π_3, and then converges. So it would query about if r_1 or r_2 is the true reward function, and then about the carpet needed to reach the switch that has a positive reward.*

8.3 Empirical Evaluations

I evaluated the algorithms in a domain similar to the one in Ch. 7, except that there are multiple switches. The domain is illustrated in Figure 8.3. It is a 6×6 robot navigation domain with 5 randomly-generated walls and different numbers of randomly-generated carpets and switches. All objects are uniformly randomly placed. The robot receives a reward of 1 by turning off one of the switches and no rewards for turning off the others. The number of reward functions is the number of switches. Each reward function indicates that the robot is rewarded by turning off a distinct switch. Each carpet corresponds to an

unknown feature. The prior belief and the probability of changeability of unknown features are randomly generated. The cost of querying is 0.1.

I generate 500 random domains based on 500 random seeds for each data point. I do not generate walls or carpets at the robot's initial position, so the robot can always follow a safe policy that stays in the initial state. Note that different from its behavior in a similar domain in Ch. 7, the robot may not end up with reaching a switch when the reward of turning off a switch does not compensate the querying cost.

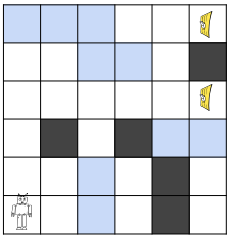


Figure 8.3: The experiment domain in this chapter.

I report results on domains with 2 switches in Figure 8.7 and 4 switches in Figure 8.8. In Figure 8.7, I find that the batch-query-based heuristic outperforms other heuristics in terms of the objective value (top-left figure), defined in Eq. 8.1. The dominating-policy-based heuristic does find safe policies with higher values than the batch-query-based heuristic, but the querying cost is much higher (top-right and bottom-left figures). The computation time of batch-query-based heuristic is more expensive compared to the others, but as the number of unknown feature increases, other heuristics also become computational expensive: the dominating-policy-based heuristic needs to compute weights (Eq. 8.2) for all dominating policies; and the myopic heuristic needs to evaluate the $EVOI$ of all unknown features before it selects a query (bottom-right figure).

Analysis of results. We have seen that, in expectation, the batch-query-based heuristic outperforms other candidate heuristics. Now I analyze when and how the batch-query-based heuristic is outperformed by any of the candidate heuristics. I experiment in a smaller domain to identify such cases. I find some cases where the batch-query-based heuristic has worse performance than the other candidate heuristics.

First, in the following example, the dominating-policy-based heuristic finds a better query policy than the batch-query-based heuristic and the myopic heuristic.

Example 8.5. *In Figure 8.4, both carpets are unknown features and have probability of 0.5 to be free. The prior reward belief is uniform. So without querying, the probability of getting a reward of 1 by turning off a switch is 0.5.*

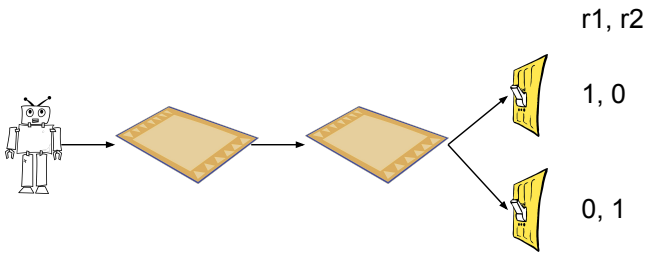


Figure 8.4: An example where the batch-query-based heuristic does not find the optimal query policy.

Clearly, there are two dominating policies. The batch-query-based heuristic first finds a policy that optimizes Eq. 8.5. Since the $EVOI$ *of either dominating policy is negative:* $0.5 * 0.5 * 0.5 * 1 - 2 * 0.1 = -0.075$ *(the robot obtains a reward of 1 if the switch it turns off has a reward (*0.5*) and both carpets are free (*$0.5 * 0.5$*); the querying cost is* $2 * 0.1$*), the robot believes that there is no policy that is worth querying about and does not pose any query.*

However, the optimal query policy first asks about the carpets. If they are both free, it poses a reward query to figure out which switch to turn off. The objective value of the optimal query policy is $-c_q + 0.5 * (-c_q + 0.5 * (-c_q + 1)) = 0.075$*, which is higher than not posing any query at all (with the objective value of 0).*

In this example, the myopic heuristic would not pose any queries either, just like the batch-query-based heuristic. The dominating-policy-based heuristic, on the other hand, would find the optimal query policy. The weight of either policy is $0.5 * 0.5 * 0.5 * (1 - 0 - 0.1 * 3) = 0.0875 > 0$*. So the dominating-policy-based heuristic would first focus on either dominating policy. It would either pose a reward query or query about its relevant features.*

In rare cases, the myopic heuristic and the dominating-policy-based heuristic may both find better query policies than the batch-query-based heuristic. The reason is the approximation I made in computing the estimated $EVOI$ of the query (Eq. 8.4). The following is an example.

Example 8.6. *In Figure 8.5, there are two carpets and two reward candidates. The prior reward belief is* $\psi = (0.9, 0.1)$*. So the switch on the left has probability 0.9 to have the reward of 1. There are two carpets and their probabilities of being free are* 0.17 *and* 0.22*.*

The batch-query-based heuristic would query about the carpet on the bottom because it is more likely to be free. If it is locked, the batch-query-based heuristic would stop querying. The estimated EVOI value is 0 if the robot keeps on querying. However, when the bottom

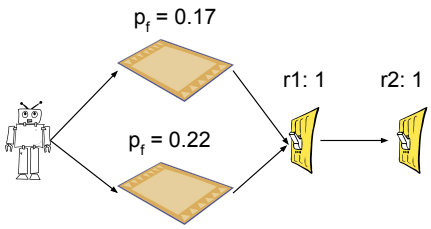


Figure 8.5: An example where the batch-query-based heuristic finds a worse query policy than batch-query-based heuristic and myopic heuristic.

carpet is locked, the myopic heuristic would then query about the carpet on the top, and if it is free, the myopic heuristic poses a reward query to decide which switch has a positive reward. the dominating-policy-based heuristic finds a similar query policy to the myopic heuristic.

The differences between the objective values of the batch-query-based heuristic and that of the myopic heuristic and the dominating-policy-based heuristic are 0.038 *and* 0.037*, respectively.*

In summary, there are two reasons why the batch-query-based heuristic fails. 1) The batch-query-based heuristic tends to overestimate the querying cost. It is computed in a way assuming that all queries need be posed at once, while by using sequential queries, some queries may not need to be posed. 2) I use a greedy construction method to find a set of policies and use that to find a query. The objective (Eq. 8.3) and the approximated objective (Eq. 8.4) are not submodular so there is no guarantee on the quality of the query policy.

Nevertheless, empirically, the batch-query-based heuristic still has better performance because it considers multiple possible dominating policies.

Large or small querying costs. I find that $c_q = 0.1$ is a setting that easily shows the differences between the different heuristics. I also ran experiments with larger and smaller querying costs. As the querying cost gets larger, myopic queries get closer to queries found by the batch-query-based heuristic. As the querying cost gets smaller, the gap between the myopic heuristic and the batch-query-based heuristic gets even larger, while the batch-query-based heuristic's performance gets closer to the dominating-policy-based heuristic.

8.4 Implementation Details: The MILP Formulation

In this section, I describe how I implement Eq. 8.5 and 8.6 as MILP problems.

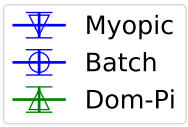


Figure 8.6: The legend for the following figures.

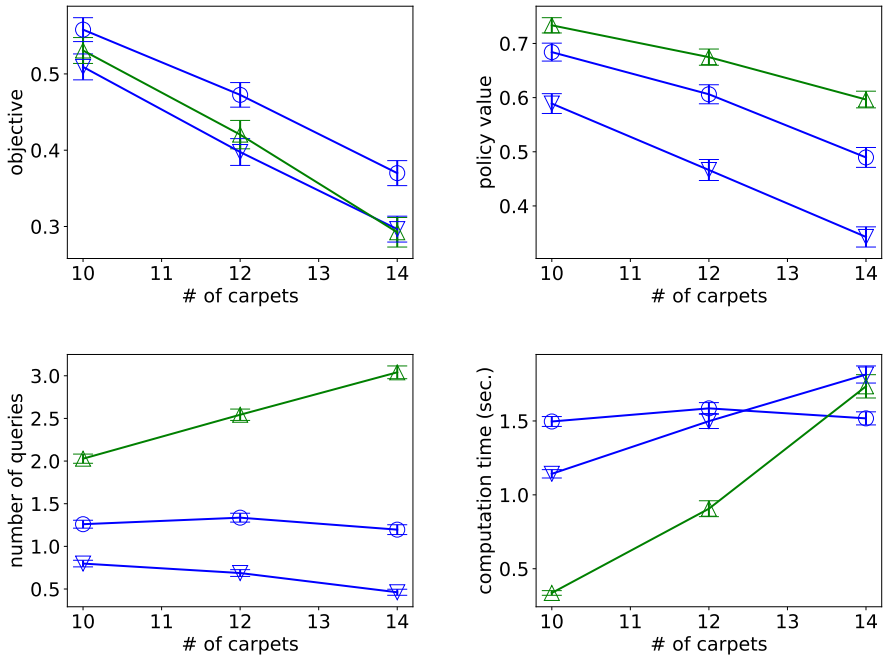


Figure 8.7: Empirical results on domains with 2 switches

Finding the first policy (Eq. 8.5). I will show that it is equivalent to the following MILP problem:

$$\max_{x,\{z_\phi\}} \sum_{s,a} x(s,a)\overline{r}(s,a) - c_q \sum_{\phi \in \Phi^{\mathbf{R}}_?} z_\phi \tag{8.7}$$

$$\text{s.t.} \sum_{a'} x(s',a') = \gamma \sum_{s,a} x(s,a)\tilde{T}(s,a,s') + 1_{[s'=s_0]},\ \forall s' \in S \tag{8.7a}$$

$$\sum_{s \in S_{\overline{\phi}}, a \in A} x(s,a) = 0,\ \forall \phi \in \Phi^{\mathbf{R}}_L \tag{8.7b}$$

$$\sum_{s \in S_{\overline{\phi}}, a \in A} x(s,a) \leq M z_\phi,\ \forall \phi \in \Phi^{\mathbf{R}}_? \tag{8.7c}$$

$$z_\phi \in \{0,1\},\ \forall \phi \in \Phi^{\mathbf{R}}_?. \tag{8.7d}$$

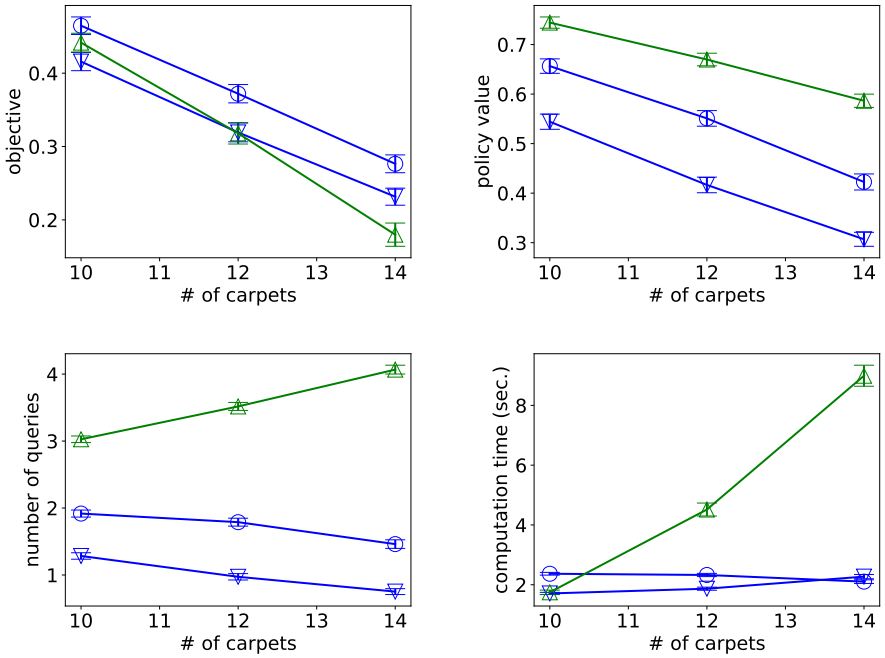


Figure 8.8: Empirical results on domains with 4 switches

The first constraint (Line 8.7a) is the flow conservation constraint that encodes the transition function, which is the same as Eq 2.9. Line 8.7b makes sure that the robot should never visit a state that changes a known-to-be-locked feature (the same as the constraint in Eq. 2.10). Line 8.7c and Line 8.7d are new constraints different from Eq. 2.8. For each unknown feature $\phi \in \Phi^{\mathbf{R}}_{?}$, I introduce an integer variable, z_ϕ (Line 8.7d). $z_\phi = 1$ indicates that the robot will query about feature ϕ and $z_\phi = 0$ otherwise. Line 8.7c is a way to encode the constraint $\sum_{s \in S_{\overline{\phi}}, a \in A} x(s,a) > 0 \implies z_\phi = 1$, that is, if the robot plans to visit states with ϕ changed, then it should query about ϕ. M is an arbitrary positive number. The robot can only visit states with changed ϕ when ϕ is known to be free. I know that this is true with probability $p_f(\phi)$. So I encode this to the transition function. Since it depends on the set of unknown features, I denote the new transition function by $\tilde{T}$.

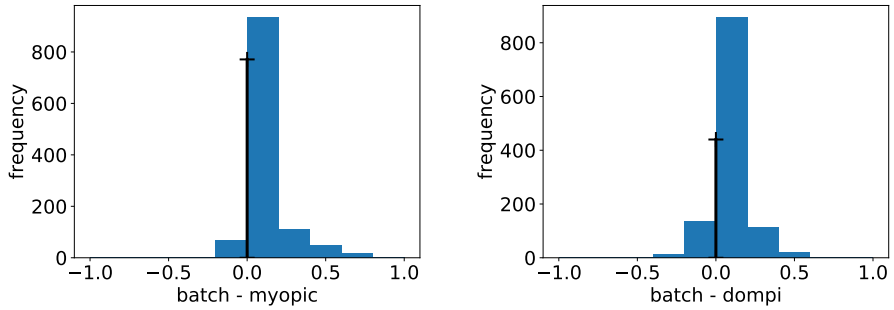


Figure 8.9: (Top) The differences of the objective values between the batch-query-based queries and the others in domains with 2 switches and 12 carpets. The vertical black segment indicates the number of trials where the difference between the two algorithms is 0.

Finding the next policy (Eq. 8.6). I will show that this objective is equivalent to the following MILP problem:

$$\max_{x,\{y_r\},\{y_r^0\},\{z_\phi\},\{z_\phi^{new}\},\{z_r\}} \sum_{r\in\mathcal{R}} \mathbb{P}[r;\psi] y_r - c_q \sum_{\phi\in\Phi_?^{\mathbf{R}}} z_\phi^{new} \tag{8.8}$$

$$\text{s.t.} \sum_{a'} x(s',a') = \gamma \sum_{s,a} x(s,a)\tilde{T}(s,a,s') + 1_{[s'=s_0]},\ \forall s' \in S \tag{8.8a}$$

$$\sum_{s\in S_{\overline{\phi}}, a\in A} x(s,a) = 0,\ \forall \phi \in \Phi_L^{\mathbf{R}} \tag{8.8b}$$

$$\sum_{s\in S_{\overline{\phi}}, a\in A} x(s,a) \le M z_\phi,\ \forall \phi \in \Phi_?^{\mathbf{R}} \tag{8.8c}$$

$$z_\phi^{new} \ge \max(z_\phi - z_\phi^0, 0),\ \forall \phi \in \Phi_?^{\mathbf{R}} \tag{8.8d}$$

$$y_r^0 = \begin{cases} V_r^{x_0}, & \text{if } \Phi_{rel}(x_0) \subseteq \Phi_{rel}(x), \\ 0, & \text{otherwise,} \end{cases} \quad \forall r \in \mathcal{R} \tag{8.8e}$$

$$y_r = \max\{V_r^x - y_r^0, 0\},\ \forall r \in \mathcal{R} \tag{8.8f}$$

$$z_\phi \in \{0,1\},\ \forall \phi \in \Phi_?^{\mathbf{R}}. \tag{8.8g}$$

In Line 8.8e, y_r^0 is the value of x^0 under reward function r if it is a safe policy and $\{\phi : z_\phi = 1 \text{ or } z_\phi^0 = 1\}$ are free features. In Line 8.8f, y_r is how much x outperforms x^0 under reward function r.

Note that Line 8.8e and 8.8f are presented in this way for clarity. They are not linear constraints. I observe that they can be written as the following linear constraints. Line 8.8e is implemented as the following. Note that the condition $\Phi_{rel}(x_0) \subseteq \Phi_{rel}(x)$ is equivalent

to $z_\phi = 1, \forall \phi \in \Phi_?^{\mathbf{R}} : z_\phi^0 = 1$. Using the tricks in Williams (2013),

$$\sum_{\phi \in \Phi_?^{\mathbf{R}} : z_\phi^0 = 1} z_\phi < |\{\phi \in \Phi_?^{\mathbf{R}} : z_\phi^0 = 1\}| + z_{safe} M \tag{8.9a}$$

$$y_r^0 \geq V_r^{x_0} - (1 - z_{safe})M, \ \forall r \in \mathcal{R} \tag{8.9b}$$

$$y_r^0 \geq 0, \ \forall r \in \mathcal{R} \tag{8.9c}$$

$$z_{safe} \in \{0, 1\} \tag{8.9d}$$

where $z_{safe} = 1$ indicates that x_0 is a safe policy under $\{z_\phi\}$.

Line 8.8f is implemented as the following.

$$y_r \leq V_r^x - y_r^0 + M(1 - z_r), \ \forall r \in \mathcal{R} \tag{8.10a}$$

$$y_r \leq 0 + M z_r, \ \forall r \in \mathcal{R} \tag{8.10b}$$

$$z_r \in \{0, 1\}, \ \forall r \in \mathcal{R} \tag{8.10c}$$

8.5 Conclusion

In this chapter, I considered the setting where both reward uncertainty and safety-constraint uncertainty are present. I showed that some straightforward heuristics may not achieve good performance. I provided a batch-query-based heuristic that is better in expectation. I also analyzed some cases where the batch-query-based heuristic finds worse query policies than do other heuristics.

For simplicity, I do not consider query languages that can simultaneously elicit both kinds of information. One example of querying to elicit both kinds of information is the following. If the robot queries about which trajectory the human likes the most in a set of trajectories, the human user may prefer one trajectory because it has a higher value, or because the other trajectories are unsafe. Although the robot is able to update its belief over the reward candidates and the partition of features at the same time, querying in this way also makes the posterior belief more uncertain since the robot is uncertain about whether the human prefers a trajectory because of the rewards or the safety constraints. So I only considered queries that reduce one type of uncertainty and leave queries that affect beliefs under both types of uncertainty for future work.

I find that using a batch-query-based heuristic can find useful information to decide what single query should be posed. To avoid evaluating all subsets of possible queries to form a good batch query, I made an approximation in the batch-query-selection objective (Eq. 8.4). Such an approximation has some drawbacks as we see in Example 8.5 and 8.6.

It would be of interest to investigate the use of an optimality guarantee on the batch-query-based heuristic, and explore the possibility of a better approximation without making the computation much more expensive.

CHAPTER 9

Conclusion

In this chapter, I conclude this dissertation by first summarizing the contributions of this dissertation and then discussing possible future work.

9.1 Summary of Contributions

Contribution I. In Ch. 5, the robot only has reward uncertainty. I develop an efficient query selection algorithm to find k-policy queries with a provable optimality guarantee, which is the first query selection algorithm in reward-uncertain MDPs with an optimality guarantee to the best of my knowledge. This work exploits the connection between the query selection objective, submodular optimization, and mixed-integer linear programming. Moreover, this algorithm is designed for finding policy queries, which may be less usable than other forms of queries, like trajectory queries. I devise an algorithm that projects a set of policies into the trajectory query space, which is more easily understandable by a human user.

This work contributes to finding high-quality queries with a budget on the number of queries that can be posed under reward uncertainty. When a robot is uncertain about the human's preferences and the human only has the patience or capacity to respond to a multiple-choice question, my algorithm can find a query with an optimality guarantee. Although the main result is about finding an approximately-optimal policy query, the projection method can help to find an askable query that contains similar information to the approximately-optimal policy query.

Contribution II. In Ch. 6, the robot only has safety-constraint uncertainty. I first formulate safety constraints as changeability of features in factored MDPs. Then I contribute to providing query selection algorithms under two scenarios. First, if the robot initially knows a safe policy, it can pose a k-feature query to find a better safe policy. We could

evaluate all k-feature queries. But I make it more efficient by pruning some known-to-be-suboptimal queries by exploiting the structure of the objective. Second, if the robot initially does not know any safe policy, it can pose sequential feature queries until it finds a safe policy or proves that no safe policy exists (Ch. 7). The goal is to reach either outcome using the minimum number of queries in expectation. I made the novel connection between the query selection objective and a pair of set cover problems, and developed an algorithm that finds better queries than other tractable baseline methods.

This work is useful when the task is safety-sensitive. Although the robot may not find the exact optimal policy, the human user still wants the robot to guarantee that its behavior is safe. The robot can query to either find a safe policy (if one exists) or prove that no safe policy exists. If the robot knows a safe policy and still has a chance to query, it may use querying to find a safe policy with higher value by confirming that the human does not care if it violates some possible safety constraints.

Contribution III. The robot may have both types of uncertainty (Ch. 8). The robot can either pose reward queries to resolve reward uncertainty or feature queries to resolve safety-constraint uncertainty. The query selection problem can be more challenging since the robot needs to decide what type of uncertainty it needs to resolve and then what query to pose. I provided two baseline query selection heuristics that combine the query selection heuristics designed for either reward uncertainty or safety-constraint uncertainty. These algorithms have clear drawbacks. I contribute a new batch-query-based heuristic that uses an imaginary batch query to guide query selection, which finds better query policies. This work provides a way to handle the scenario where both types of uncertainty are present, which is more realistic in the real world.

9.2 Future Work

Optimality guarantees. I provided optimality guarantees on the quality of queries found by the algorithms in Ch. 5 and 6. It would be of interest to provide optimality guarantees for the query selection algorithms in Ch. 7 and 8. The query selection algorithm that finds an initial safe policy (Ch. 7) is motivated by set cover algorithms, which have optimality guarantees on set cover problems. Unfortunately, the optimality guarantee cannot be directly applied to my query selection algorithm since my algorithm aims to solve two set cover problems simultaneously. The query selection algorithm under both types of uncertainty does not have an optimality guarantee. I only show analytically why it is better than some baseline heuristics. In general, it is challenging to provide optimality guarantees when we have sequential queries.

The main difficulty of these problems is that the objective is usually no longer submodular and there is no clear way to prune the query space. It would be of interest to find useful tools to analyze the optimality of these query selection algorithms or inspire new query selection algorithms.

Scalability to large MDPs. The algorithms provided in this thesis are designed for MDPs with a small number of states. I hope to generalize these algorithms to domains with larger or continuous state spaces. Unfortunately, it is more difficult to provide theoretical guarantees on query selection algorithms in continuous-state MDPs. Under only reward uncertainty, for example, finding an optimal reward query would require finding optimal policies under a reward belief, while it is already impractical to find an optimal policy given a reward function. So most query selection algorithms in the literature do not have an optimality guarantee. An example of finding constrained optimal policy using neural networks is Christiano et al. (2017).

Although it is difficult to provide approximate optimality guarantees when we find approximate algorithms on large domains, fortunately, some other guarantees are not lost. In Ch. 6 and 7, we may not be able to find all dominating policies in a large domain, but we can find queries using a subset of dominating policies and still guarantee that the robot's behavior is safe. The quality of the query may be compromised, but the safety guarantee still holds. Also, the design of my algorithms does not prohibit extensions to large domains. I mostly use linear programming or mixed integer linear programming formulations to find optimal or safely-optimal queries. These components can be replaced by neural networks, which are suitable for large domains (Achiam et al. 2017).

A more embodied query selection algorithm. In this thesis, each chapter considers a different query selection objective under different uncertainty settings. For example, under safety-constraint uncertainty, I use the algorithm in Ch. 6 when an initial safe policy is known, and the algorithm in Ch. 7 when no initial safe policy is known. Ch. 8 shows a way to handle both reward uncertainty and safety-constraint uncertainty, However, the best algorithm under both types of uncertainty (the batch-query-based heuristic) does not inherit the optimality guarantees of algorithms designed for one type of uncertainty. An interesting future direction would be one query selection algorithm that can elegantly handle different types of uncertainty.

9.3 Summary

To summarize, this thesis makes the contribution of designing query selection algorithms under reward uncertainty and safety-constraint uncertainty. These algorithms either have optimality guarantees or empirical better performance than the baseline methods.

As we can see in these settings, optimal query selection can be challenging. Most works in the literature focus on using heuristics that find empirically good queries in certain domains. My algorithms provide optimality by exploiting some structure in the query selection objective: When the objective function is submodular, I use a greedy construction approach to find approximately-optimal queries. Otherwise, I examine the query space to prune the queries that are known to be suboptimal.

This thesis builds and exploits a connection between the human-agent interaction literature and approximate algorithm literature. For example, in Ch. 5, I am able to provide an optimality guarantee to the query selection algorithm since the objective is submodular. In Ch. 7, I provide a novel formulation of the querying problem as a pair of set cover problems. More generally, this thesis contributes to solving the reward design problem. In reality, the robot would not receive a completely-defined reward function before it is assigned to a task. The robot may communicate with the human about what the human wants and such communication would not last arbitrarily long. My query selection algorithms can be used to identify the human's objectives and make sure that the robot's behavior does not negatively surprise the human.

I hope this thesis can inspire research on querying under uncertainty in the following ways. First, most query selection methods in the literature are heuristic-based without optimality guarantees. I provide ways of proving optimality guarantees on query selection methods, which may be generalized to other problems or inspire other provably-optimal algorithms. Second, query selection problems in larger domains can be challenging. Although the algorithms in this thesis focus on domains with a small number of states, we can find approximations to these algorithms so that they can be applied in large or continuous domains. Finally, I formulate the robot's uncertainty as reward uncertainty and safety-constraint uncertainty, and I do not claim that these two types of uncertainty can fully capture the robot's uncertainty. We can explore other types of uncertainty and see if the query selection algorithms in this thesis can be generalized to different settings.

BIBLIOGRAPHY

BIBLIOGRAPHY

Abbeel, Pieter and Andrew Y Ng (2004). "Apprenticeship learning via inverse reinforcement learning". In: *Proceedings of the Twenty-First International Conference on Machine learning*, pp. 1–8.

Achiam, Joshua, David Held, Aviv Tamar, and Pieter Abbeel (2017). "Constrained policy optimization". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 22–31.

Akrour, Riad, Marc Schoenauer, and Michele Sebag (2012). "APRIL: active preference learning-based reinforcement learning". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 116–131.

Akrour, Riad, Marc Schoenauer, and Michele Sebag (2013). "Interactive robot education". In: *ECML/PKDD Workshop on Reinforcement Learning with Generalized Feedback: Beyond Numeric Rewards*.

Alshiekh, Mohammed, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu (2018). "Safe Reinforcement Learning via Shielding". en. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. (Visited on 04/17/2019).

Altman, Eitan (1999). *Constrained Markov Decision Processes*. Vol. 7. CRC Press.

Amin, Kareem, Nan Jiang, and Satinder Singh (2017). "Repeated inverse reinforcement learning". In: *Adv. in Neural Info. Proc. Sys. (NIPS)*, pp. 1813–1822.

Amodei, Dario, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané (2016). "Concrete problems in AI safety". In: *arXiv preprint arXiv:1606.06565*.

Boutilier, Craig, Thomas Dean, and Steve Hanks (1999). "Decision-theoretic planning: Structural assumptions and computational leverage". In: *J. of Artificial Intelligence Research (JAIR)* 11.1, p. 94.

Chaslot, Guillaume (July 2019). "The Toxic Potential of YouTube's Feedback Loop". In: *Wired*. ISSN: 1059-1028. URL: https://www.wired.com/story/the-toxic-potential-of-youtubes-feedback-loop/ (visited on 03/22/2020).

Chinneck, John W. (2007). *Feasibility and Infeasibility in Optimization:: Algorithms and Computational Methods*. Vol. 118. Springer Science & Business Media.

Chow, Yinlam, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone (2017). "Risk-constrained reinforcement learning with percentile risk criteria". In: *The Journal of Machine Learning Research* 18.1. Publisher: JMLR. org, pp. 6070–6120.

Christiano, Paul F., Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei (2017). "Deep reinforcement learning from human preferences". In: *Advances in Neural Information Processing Systems*, pp. 4299–4307.

Clark, Jack and Dario Amodei (Dec. 2016). *Faulty Reward Functions in the Wild*. Library Catalog: openai.com. URL: https://openai.com/blog/faulty-reward-functions/ (visited on 03/19/2020).

Cohn, R., M. Maxim, E. Durfee, and S. Singh (2010). "Selecting operator queries using expected myopic gain". In: *ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pp. 40–47.

Cohn, Robert (2016). "Maximizing Expected Value of Information in Decision Problems by Querying on a Wish-to-Know Basis". PhD Thesis. University of Michigan.

Cohn, Robert, Edmund H. Durfee, and Satinder Singh (2011). "Comparing action-query strategies in semi-autonomous agents". In: *Int. Conf. on Autonomous Agents and Multiagent Systems*, pp. 1287–1288.

Cohn, Robert, Satinder Singh, and Edmund Durfee (2014). "Characterizing EVOI-sufficient k-response query sets in decision problems". In: *Conference on Artificial Intelligence and Statistics*, pp. 131–139.

Farias, Daniela Pucci de and Benjamin Van Roy (2003). "The linear programming approach to approximate dynamic programming". In: *Operations Research* 51.6, pp. 850–865.

Garcıa, Javier and Fernando Fernández (2015). "A comprehensive survey on safe reinforcement learning". In: *J. of Machine Learning Research (JMLR)* 16.1, pp. 1437–1480.

Golovin, D. and A. Krause (2011). "Adaptive Submodularity: Theory and Applications in Active Learning and Stochastic Optimization". en. In: *Journal of Artificial Intelligence Research* 42, pp. 427–486.

Goodrich, Michael A and Alan C Schultz (2007). "Human-robot interaction: A survey". In: *Foundations and trends in human-computer interaction* 1.3, pp. 203–275.

Hadfield-Menell, Dylan, Anca Dragan, Pieter Abbeel, and Stuart Russell (2016a). "The Off-Switch Game". In: *arXiv preprint arXiv:1611.08219*.

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

Hadfield-Menell, Dylan, Stuart J Russell, Pieter Abbeel, and Anca Dragan (2016b). "Cooperative inverse reinforcement learning". In: *Advances in Neural Information Processing Systems*, pp. 3909–3917.

Jain, Ashesh, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena (2013). "Learning trajectory preferences for manipulators via iterative improvement". In: *Advances in Neural Information Processing Systems*, pp. 575–583.

Knox, W. Bradley and Peter Stone (2009). "Interactively Shaping Agents via Human Reinforcement: The TAMER Framework". In: *Proceedings of the Fifth International Conference on Knowledge Capture*. K-CAP '09. New York, NY, USA, pp. 9–16.

Kolobov, Andrey, Mausam, and Daniel S. Weld (2012). "A Theory of Goal-Oriented MDPs with Dead Ends". In: *Proc. Conf. on Uncertainty in Artificial Intelligence (UAI)*, pp. 438–447.

Krakovna, Victoria, Laurent Orseau, Miljan Martic, and Shane Legg (2018). "Penalizing Side Effects using Stepwise Relative Reachability". In: *AISafety@IJCAI*.

Krause, Andreas and Daniel Golovin (2014). "Submodular Function Maximization". In: *Tractability*. DOI: 10.1017/CBO9781139177801.004.

Le, Tiep, Atena M. Tabakhi, Long Tran-Thanh, William Yeoh, and Tran Cao Son (2018). "Preference Elicitation with Interdependency and User Bother Cost". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1459–1467. URL: http://dl.acm.org/citation.cfm?id=3237383.3237918 (visited on 07/23/2019).

Leike, Jan, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg (2018). "Scalable agent alignment via reward modeling: a research direction". In: *arXiv preprint arXiv:1811.07871*.

Leike, Jan, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg (2017). "AI safety gridworlds". In: *arXiv preprint arXiv:1711.09883*.

Mark, Gloria, Daniela Gudith, and Ulrich Klocke (2008). "The cost of interrupted work: more speed and stress". In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 107–110.

Mark, Gloria, Shamsi T. Iqbal, Mary Czerwinski, Paul Johns, Akane Sano, and Yuliya Lutchyn (2016). "Email duration, batching and self-interruption: Patterns of email use on productivity and stress". In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pp. 1717–1728.

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

Mindermann, Sören, Rohin Shah, Adam Gleave, and Dylan Hadfield-Menell (Sept. 2018). "Active Inverse Reward Design". In: *arXiv:1809.03060 [cs, stat]*. arXiv: 1809.03060. URL: http://arxiv.org/abs/1809.03060 (visited on 03/26/2019).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602*.

Palan, Malayandi, Nicholas C. Landolfi, Gleb Shevchuk, and Dorsa Sadigh (2019). "Learning Reward Functions by Integrating Human Demonstrations and Preferences". In: *Robotics: Science and Systems*.

Ramachandran, Deepak and Eyal Amir (2007). "Bayesian inverse reinforcement learning". In: *International Joint Conference on Artificial Intelligence*, pp. 2586–2591.

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

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

Regan, Kevin and Craig Boutilier (2011a). "Eliciting additive reward functions for Markov decision processes". In: *International Joint Conference on Artificial Intelligence*. Vol. 22, pp. 2159–2164.

Regan, Kevin and Craig Boutilier (2011b). "Robust online optimization of reward-uncertain MDPs". In: *International Joint Conference on Artificial Intelligence*. Vol. 22, pp. 2165–2171.

Sadigh, Dorsa, S. Shankar Sastry, Sanjit A. Seshia, and Anca Dragan (2016). "Information gathering actions over human internal state". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 66–73.

Shah, Rohin, Dmitrii Krasheninnikov, Jordan Alexander, Pieter Abbeel, and Anca D. Dragan (2019). "Preferences Implicit in the State of the World". In: *International Conference on Learning Representations*.

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, and Marc Lanctot (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587. Publisher: Nature Publishing Group, p. 484.

Smith, Trey and Reid Simmons (2004). "Heuristic search value iteration for POMDPs". In: *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pp. 520–527.

Sutton, Richard S. and Andrew G. Barto (Oct. 2018). *Reinforcement Learning: An Introduction*. en. MIT Press. ISBN: 978-0-262-35270-3.

Teichteil-Königsbuch, Florent (2012). "Stochastic Safest and Shortest Path Problems". In: *Proc. Assoc. for Adv. of Artificial Intelligence (AAAI)*, pp. 1825–1831.

Thomason, Jesse, Shiqi Zhang, Raymond J. Mooney, and Peter Stone (2015). "Learning to interpret natural language commands through human-robot dialog". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*, pp. 1923–1929.

Torrey, Lisa and Matthew Taylor (2013). "Teaching on a budget: Agents advising agents in reinforcement learning". In: *International Conference on Autonomous Agents and Multi-agent Systems*, pp. 1053–1060.

Turner, Alexander Matt, Dylan Hadfield-Menell, and Prasad Tadepalli (2020). "Conservative agency via attainable utility preservation". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pp. 385–391.

Viappiani, Paolo and Craig Boutilier (2009). "Regret-based optimal recommendation sets in conversational recommender systems". In: *Proc. ACM Conf. on Recommender Systems*, pp. 101–108.

Viappiani, Paolo and Craig Boutilier (2010). "Optimal Bayesian recommendation sets and myopically optimal choice query sets". In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 2352–2360.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning". In: *Machine learning* 8.3-4. Publisher: Springer, pp. 279–292.

Williams, H. Paul (2013). *Model building in mathematical programming*. John Wiley & Sons.

Williamson, David P. and David B. Shmoys (Apr. 2011). *The Design of Approximation Algorithms*. Cambridge University Press. ISBN: 978-1-139-49817-3.

Wilson, Aaron, Alan Fern, and Prasad Tadepalli (2012). "A Bayesian approach for policy learning from trajectory preference queries". In: *Advances in Neural Information Processing Systems*, pp. 1133–1141.

Witwicki, Stefan J and Edmund H. Durfee (2010). "Influence-Based Policy Abstraction for Weakly-Coupled Dec-POMDPs". In: *Proc. Int. Conf. Auto. Planning and Scheduling (ICAPS)*, pp. 185–192.

Zhang, Qi, Edmund H. Durfee, and Satinder Singh (2020a). "Semantics and algorithms for trustworthy commitment achievement under model uncertainty". In: *Autonomous Agents and Multi-Agent Systems* 34.1. Publisher: Springer, pp. 19–35.

Zhang, Shun, Edmund H. Durfee, and Satinder Singh (2017). "Approximately-optimal queries for planning in reward-uncertain Markov decision processes". In: *Proceedings*

of the 27th International Conference on Automated Planning and Scheduling (ICAPS), pp. 339–347.

Zhang, Shun, Edmund H. Durfee, and Satinder Singh (2018). "Minimax-regret querying on side effects for safe optimality in factored Markov decision processes". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4867–4873.

Zhang, Shun, Edmund H. Durfee, and Satinder Singh (2020b). "Querying to Find a Safe Policy Under Uncertain Safety Constraints in Markov Decision Processes". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.