REGRET-BASED REWARD ELICITATION FOR MARKOV DECISION PROCESSES

by

Kevin Regan

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Regret-Based Reward Elicitation

for Markov Decision Processes

Kevin Regan

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2014

Markov decision processes (MDPs) have proven to be a useful model for sequential decision-theoretic reasoning under uncertainty, yet they require the specification of a reward function that can require sophisticated human judgement to assess relevant tradeoffs. This dissertation casts the problem of specifying rewards as one of preference elicitation and aims to minimize the degree of precision with which a reward function must be specified while still allowing optimal or near-optimal policies to be produced. We demonstrate how robust policies can be computed for MDPs given only partial reward information using the minimax regret criterion.

Minimax regret offers an intuitive bound on loss; however, it is computationally intractable in general. This work develops techniques for exploiting MDP structure to allow for offline precomputation that enables efficient online minimax regret computation. To complement this exact approach we develop several general approximations that offer both upper and lower bounds on minimax regret. We further show how approximations can be improved *online* during the elicitation procedure to balance accuracy and efficiency.

To effectively reduce regret, we investigate a spectrum of elicitation approaches that range from the computationally-demanding optimal selection of complex queries about full MDP policies (which are informative, but, we believe, cognitively difficult) to the heuristic selection of simple queries that focus on a small set of reward parameters. Results are demonstrated on MDPs drawn from the domains of assistive technology and autonomic computing.

Finally we demonstrate our framework on a realistic website optimization domain, performing elicitation on websites with tens of thousands of webpages. We show that minimax regret can be efficiently computed, and develop informative and cognitively reasonable queries that quickly lower minimax regret, producing policies that offer significant improvement in the design of the underlying websites.

Acknowledgements

I would like to thank my advisor Craig Boutilier for the deep insight and thoughtful guidance he offered on my research topic. I would like to thank my committee members Fahiem Bacchus, Sheila McIlraith, Richard Zemel and my external examiner Leslie Pack Kaelbling for their time and for their constructive feedback.

I am grateful to my fellow students at the University of Toronto for their camaraderie and counsel, and to the staff for making all the bureaucratic details of navigating graduate school as painless as possible.

I would also like to thank my family: my parents Peggy and Joe, my sister Michelle, and my partner Elaine for their incredible love and support. Finally, I want to thank my son Phineas for the inspiration he provided near the end of my long journey.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Sequential decision-theoretic reasoning under uncertainty is a general and important problem in both operations research and artificial intelligence. The aim is to construct policies that dictate optimal actions given the current state of the world and a model quantifying the impact of actions on the future state of the world. Policies are chosen to optimize a reward function that captures the desirability of encountering a particular state and taking a particular action.

Many real-world reasoning problems admit description in these terms. For instance, planning and logistics at large organizations often involve optimizing processes with series of actions with stochastic effects. These processes range from shipping and component sourcing to "virtual" resource allocation in computing infrastructure—known as *autonomic computing*. Another example is autonomous vehicles that must optimize their operation in the face of uncertainty, weighing the risk inherent in a course of action with the reward obtained for achieving desired goals. Semi-autonomous systems interacting directly with people can be modeled similarly; a cognitive assistance system that helps persons with dementia to navigate daily tasks must take input from noisy sensors and plan a series of actions that trade-off the person's autonomy with the goal of task completion. The design of websites can be cast as sequential decision problem that finds a series of actions specifying the webpage to serve in response to an HTTP request in order to optimize a visitor's (uncertain) behaviour toward some goals such as the purchase of a product.

Markov decision processes (MDPs) have proven to be a useful model sequential decision-theoretic reasoning under uncertainty, yet they require the specification of a large number of parameters to capture both the stochastic dynamics of action effects along with the reward function. While dynamics can be learned by observation of the environment, the reward function reflects the subjective preferences of some user and can require sophisticated human judgement to assess relevant tradeoffs. For instance, when designing a website, stochastic user behaviour can easily be captured by logging activity on each webpage. However, precisely specifying trade-offs between goals like user-engagement and conversion can be difficult.

In some cases there are a simple set of goals/preferences that map directly to observations in the environment. However, in general the specification of rewards is problematic, since it requires the translation of general user preferences into *precise* quantities—a complex and cognitively demanding task, well documented in the decision theory literature (Keeney and Raiffa, 1976). Furthermore, this time-consuming process may need to be repeated to capture the varying preferences of different users (Slavic, Fischhaff, and Lichtenstein, 1977).

An analog to reward specification can be found in the task of determining utility functions in single-step decision problems. An example of such a problem is the decision support task of determining a user's utility for potential car purchases. There is an important trade-off between the improvement in quality of the decision made possible through eliciting utility information and the burden associated with eliciting that information. In some cases this burden takes the form of time or mental effort expended by the user; in other cases, computational resources that are consumed to run simulations (to gauge the utility of particular configurations in the autonomic computing domain, for example). The improvement in decision quality offered by eliciting further reward information does not always justify the burden imposed (Salo and Hämäläinen, 1995; Chajewska, Koller, and Parr, 2000; Wang and Boutilier, 2003; Boutilier, Das, Kephart, Tesauro, and Walsh, 2003a; Boutilier, Patrascu, Poupart, and Schuurmans, 2006), motivating methods for making decisions with partial preference information. In the single-step car recommendation task, a user's preferences for inexpensive cars may obviate the need to specify the utility of costly luxury options. This phenomenon extends to sequential problems, where the dynamics of an MDP may dictate that certain states are rarely visited under any policy of action; omitting the precise specification of reward for such states will have relatively little impact on expected total reward of a chosen policy.

In some cases reward specification may be assisted by a priori information in the form of a full probability distribution over possible rewards. We wish to avoid relying on such assumptions and instead assume only that the unknown reward function is one of a (possibly) infinite set of possible reward functions. By not being required to construct and maintain a precise probabilistic model of the unknown reward function, we side-step the potentially intractable inference required to update the model during elicitation—though we will see that some assumptions we later adopt yield computational challenges of their own. Given *strict*, *set-based* uncertainty over the reward function we can apply robustness criteria in order to compute policies that offer tight guarantees on policy value given any instantiation of reward.

Some often cited examples of robustness criteria are the maximin and maximax criteria, which dictate policies that are respectively optimal w.r.t. worst-case and best-case realizations of the reward function. While both offer guarantees, the criteria suffer from either being too optimistic (maximax) or too pessimistic (maximin). The minimax regret criterion (Savage, 1954) suggests a more reasonable measure, selecting a policy that minimizes maximum regret (or reward loss). A policy is thus measured not with respect to the worst-case reward in isolation, but instead a policy is measured against the best policy that could have been selected w.r.t. a realized reward. Thus terrible worst-case rewards that penalize all policies equally are effectively ignored. Minimax regret bounds the amount of additional reward that could be gained if the reward function were fully specified; this bound serves as a natural measure for deciding whether to undertake the burden of further elicitation.

If minimax regret is shown to be zero, then the resulting policy is provably optimal and no further elicitation is required. For these reasons the minimax regret criterion has been adopted

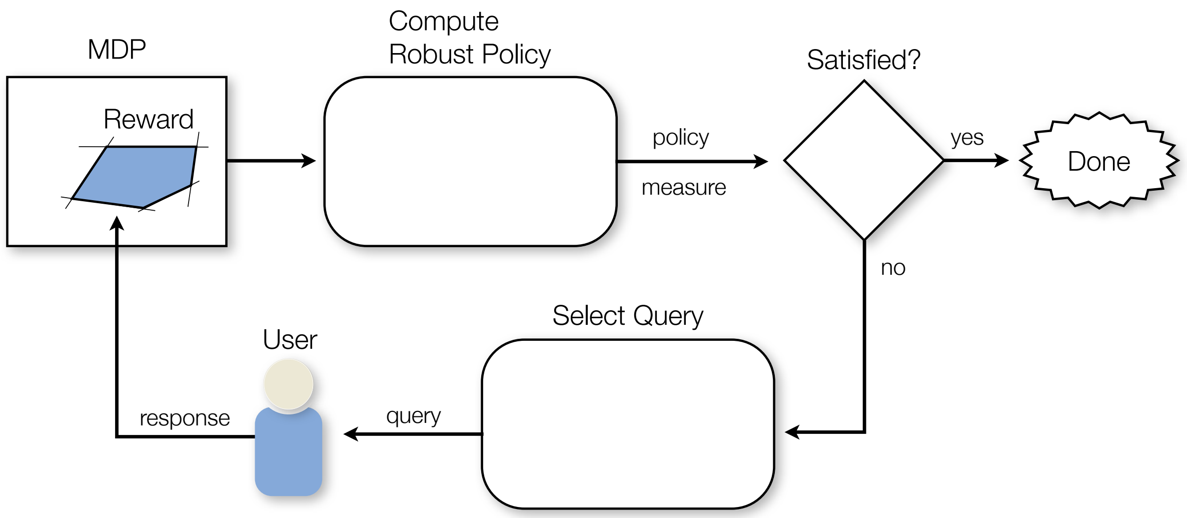


Figure 1.1: Diagram of reward elicitation steps.

as a natural and intuitive criterion in a number of settings (Boutilier et al., 2006; Salo and Hämäläinen, 1995; Xu and Mannor, 2009).

Given a particular robust criterion, the choice of policy can be improved by further specifying the unknown reward function through elicitation. Additional reward information can be passively gathered by observing user activity. For instance, human care-givers could be monitored to glean reward information in the cognitive assistance domain. The subfield of inverse reinforcement learning describes approaches for inferring reward function constraints from the demonstrated behaviour of users (Ng and Russell, 2000). The related theory of *revealed preference* in economics holds that a user does not possess a weak preference ordering a priori, but rather that a preference relation is constructed (or revealed) through observing choices made by the user (Samuelson, 1948).

In this work we focus on "active" elicitation that directly engages the user with queries about their preferences. Active elicitation can be viewed as an iterative procedure: at each step a policy is recommended along with a measure of its "goodness" w.r.t. to a robustness criterion; if the user is unsatisfied, a query is selected and posed to the user; the response further circumscribes the set of feasible rewards, potentially improving the recommended policy. This process is visualized in Figure 1.1. The focus of query selection here is not to lower reward uncertainty for its own sake, but rather to directly improve the recommended policy w.r.t. to the criterion. In fact, information generated from the current application of the criterion can guide query selection to outperform methods based solely on the characteristics of the set of feasible rewards (Boutilier et al., 2006). Techniques for incrementally eliciting utility functions for single step decision problems using minimax regret have proven effective in allowing optimal or near-optimal decisions to be found without full utility specification (Boutilier et al., 2006) and provide the inspiration our work.

We adopt a model for sequential decision making with partially specified reward functions (McMahan, Gordon, and Blum, 2003; Regan and Boutilier, 2010; Xu and Mannor, 2009)), replacing the reward function with a set of feasible reward functions (assuming strict uncertainty). We refer to this extended model as the *imprecise reward MDP* (IRMDP).

> The central aim of this thesis is the development of a minimax regret-based framework for the incremental elicitation of reward functions for IRMDPs that lowers the burden on users while remaining computationally effective.

1.1 Challenges

The first major challenge we face is developing an effective approach to minimax regret computation for IRMDPs, since it is computationally intractable in general (Xu and Mannor, 2009). A suitable approach must facilitate real-time interactive elicitation with a user, tackling large "real-world" IRMDPs and delivering results in seconds rather than minutes or hours.

This objective is achieved through exploiting structure to allow for offline precomputation that enables efficient online computation. For any IRMDP there exists a set of polices that are *dominated* w.r.t. possible reward realizations. These dominated policies can be safely ignored, and a precomputed set of nondominated policies can be leveraged to dramatically improve the efficiency of computing minimax regret during elicitation. Specifically, the minimax regret computation scales linearly with the number of nondominated policies. We develop a polynomial time algorithm—the πWitness algorithm—to identify the set of nondominated policies and show that for IRMDPs admitting a polynomial number of nondominated policies, we can compute minimax regret in polynomial time. Most promising of our approaches to precomputing nondominated policies is an anytime algorithm—the nondominated-region vertex (NRV) algorithm—that offers a bound on the approximation error that would be incurred at each step by using the currently identified subset of nondominated policies. As new nondominated policies are identified, the runtime of the minimax regret approximation using the current set of nondominated policies increases while the approximation error decreases; selecting the point at which to terminate the anytime algorithm (NRV) allows for precise trade-offs between approximation error and efficiency.

Some parameterizations of the IRMDP—such as the uncalibrated additive reward functions explored in Chapter 5—do not lend themselves to efficient nondominated policy generation. For these cases we develop several approximations that do not rely on nondominated polices. These approximations offer both upper and lower bounds on minimax regret; and can be used to both guide query selection and provide a bound on the regret of the current policy during elicitation, thereby providing a suitable stopping criterion.

The next major challenge is the reduction of minimax regret through effective elicitation. Our work investigates a spectrum of elicitation approaches that range from the computationally-demanding optimal selection of complex queries about full policies (which are informative, but, we believe, cognitively difficult) to the heuristic selection of simple queries that focus on a small set of reward parameters. We empirically examine how query structure and query parameter selection influence the effectiveness of the resulting reward elicitation procedure, focusing on example IRMDPs drawn from the domains of assistive technology and autonomic computing.

We show how a factored state space and structural independence in the reward function can be leveraged to pose more direct (and intuitive) queries that focus on individual attributes of the reward function and streamline elicitation. The elicitation of rewards for large factored IRMDPs can thus be accomplished with a small number of targeted, intuitive queries.

Effective elicitation can in turn improve the efficiency of minimax regret computation. An approximate set of nondominated policies can be carefully managed during elicitation to compute approximate minimax regret with bounded error—error that is reduced as elicitation proceeds and reward uncertainty is reduced. As reward uncertainty is reduced during elicitation, nondominated policies in this set will become dominated, and can safely be removed, improving the efficiency of minimax regret computation (which depends heavily on the number of nondominated policies). Additional nondominated policies can then be added to the approximate set lowering error. This allows us to make explicit tradeoffs between the quality of approximation and the efficiency of computation.

Finally, we simultaneously tackle the twin challenges of minimax regret computation and effective reward elicitation for a realistic IRMDPs by tackling a website optimization decision problem. We build models of user behaviour from four websites, each with tens of thousands of webpages. We show that minimax regret can be efficiently computed, and develop informative and cognitively reasonable queries and use these queries to quickly lower minimax regret, producing policies that offer significant improvement in the design of the underlying websites.

1.2 Contributions

To summarize the novel aspects of this work we list our major contributions below.

Computing Robust Policies using Minimax Regret (MMR)

- An exact procedure for MMR computation using constraint generation and mixed integer programming
- Several approximate methods for efficiently generating lower and upper bounds on MMR
- An exact approach to minimax regret computation leveraging nondominated policies
- A polynomial time algorithm (πWitness) for generating nondominated policies

Reward Elicitation

- The development of of volumetric and current solution heuristics for query selection
- A method for optimal full-policy query selection

Leveraging Reward Structure

- Exact and approximate algorithms computing MMR for IRMDPs with additive reward
- Decision-theoretically sound heuristics for eliciting additive reward using local value functions

Online Minimax Regret Computation

- The *nondominated region vertex* (NRV) algorithm for generating nondominated policies which provides an anytime bound on approximation error for minimax regret
- A method for adjusting the set of nondominated policies online, speeding up computation and improving the quality of approximation

Reward Elicitation for Website Optimization

- The empirical analysis of public datasets demonstrating effective version testing optimization on websites with thousands of pages.
- Examples of effective and cognitively reasonable full policy queries

Each of these contributions will be placed in the context of related work as our methods are developed in later chapters. However, it worth noting here that our work comprises the first fully realized framework for the incremental elicitation of reward for Markov decision processes.

1.3 Outline

This thesis is structured as follows. Chapter 2 introduces the relevant background on single-step preference elicitation and sequential decision making. Reward function elicitation for Markov decision processes can be viewed as the confluence of these two streams of research.

Chapter 3 formally specifies the IRMDP model and develops a number of approaches to computing minimax regret for IRMDPs. We first detail an exact method using constraint generation and mixed integer programming. We observe that the runtime of this approach scales superlinearly with size of IRMDPs. To address this, we detail three approximations, two of which offer lower bounds and one an upper bound on minimax regret. Next, we examine how a precomputed set of nondominated policies can be used to compute minimax regret. We complete this approach by developing an algorithm for generating the set of nondominated policies and we observe that the runtime of minimax regret is tightly linked with the number of nondominated policies. We then extend our approach to anytime nondominated policy generation (a topic that we will revisit in depth in Chapter 6).

Chapter 4 introduces our approach to reward elicitation for general IRMDPs (i.e., without assuming factored structure). We detail two heuristic approaches to query selection: the first uses information about only the feasible reward set, while the second supplements this with information from the minimax regret solution at each step of the elicitation. Next we develop a (myopically) optimal query selection method for full policy queries (which will be demonstrated on the website optimization domain in Chapter 7) and demonstrate the effectiveness of this approach.

Chapter 5 describes how reward structure can be leveraged to improve the effectiveness of elicitation. We extend our approaches to compute minimax regret for IRMDPs with additive reward composed of local value function and develop decision-theoretically sound heuristics to elicit information about these local value functions and along with parameters specifying their global calibration.

Chapter 6 revisits the online use of nondominated policies. Shrewd management of an approximate set of nondominated policies during elicitation can yield efficient computation of approximate minimax regret. We develop the nondominated region vertex algorithm for anytime nondominated policy generation, which yields a bound on error and operates by generating policies to maximally reduce that error. Paired with our online management of nondominated policies, the NRV algorithm allows for approximate minimax regret computation with bounded error that is reduced as elicitation proceeds.

Chapter 7 applies many of the methods developed in Chapters 3–6 on a website optimization problem. We detail how we simulate website IRMDPs using data from four existing datasets and describe the results of reward elicitation in this domain.

Chapter 8 concludes this thesis with a summary of contributions and discussion of the many directions for future work.

Chapter 2

Background

This chapter presents background material referenced throughout the proposed thesis and provides a survey of related work.

2.1 Single-step Preference Elicitation

In this section we introduce the principles of single-step decision making and preference elicitation that, along with Markov decision processes, will form the foundation for preference elicitation in multi-stage decision processes.

2.1.1 Basic Decision Theory

Decision theory provides a framework for modeling the preferences of a user and stipulates how optimal decisions are to be made based on these preferences. A comprehensive treatment can be found in the work of von Neumann and Morgenstern (1944), Fishburn (1970), Keeney and Raifa (1976) and French (1986).

The simplest case focuses on a single user choosing among a finite set $\mathcal{X}$ of possible outcomes. The preferences of the user are specified by a set of binary relations. The *preference* relation $x \succ y$ denotes that outcome x is preferred to outcome y. The *preference* relation is asymmetric and it can not be the case that both $x \succ y$ and $y \succ x$. To capture this notion, the symmetric *indifference* relation is used: $x \sim y$ denotes the user being indifferent between x and y. A *weak preference* relation combines the two relations: $x \succeq y$ denotes that x is at least as preferred as y (either $x \succ y$ or $x \sim y$). It is generally assumed that any decision h is rational and has the following attitudes towards certain outcomes.

Comparability: All outcomes are comparable: $\forall\ x, y \in \mathcal{X}$, either $x \succ y$, $y \succ x$, or $x \sim y$.

Transitivity: All orderings of outcomes are consistent. $\forall\ x, y, z \in \mathcal{X}$, if $x \succ y$ and $y \succ z$, then $x \succ z$.

A weak preference ordering over outcomes that is comparable and transitive can be represented on the *ordinal scale* using a scalar value function that maps each outcome to a real number. On this scale the user prefers outcomes with higher values. However, given simple binary relations between outcomes, it is not possible to represent the magnitude of a preference (i.e., "By how much is x preferred to y"?).

2.1.2 Quantitative Preferences

Von Neumann and Morgenstern (1944) showed that binary preference relations can be mapped to a cardinal scale by pairing outcomes with a probability of occurrence. They introduce the notion of a lottery over outcomes. A simple lottery l is defined as the set $\langle p_1, x_1; p_2, x_2; \ldots; p_n, x_n \rangle$ in which each outcome x_i has a probability p_i of being realized. The probabilities p_i sum to one and form a distribution over the set of outcomes
$\mathcal{X} \equiv \{x_1, x_2, \ldots, x_n\}$.

The user's preferences over outcomes can be extended to include preferences over lotteries. When the preferences over lotteries obey a small set of axioms (specifically the decomposability, independence, and continuity axioms), they can be represented by a *utility function* u which maps each lottery to a real-valued *utility* such that lotteries with higher utility are preferred.

The utility of a simple outcome x can be measured using degenerate lotteries in which the outcome x occurs with certainty. It can then be shown that the utility of a lottery is equal to the expected utility of the outcomes in the lottery (von Neumann and Morgenstern, 1944).

$$u(\ l = \langle p_1, x_1; p_2, x_2; \ldots; p_n, x_n \rangle\ ) = \sum_i^n p_i u(x_i) \tag{2.1}$$

It follows that when an action has uncertainty as to its outcomes, a rational user should choose the action that maximizes the expected utility of the outcomes.

2.1.3 Multi-Attribute Utility

It is often natural to consider outcomes as having a multi-dimensional structure. For instance when choosing a car to buy, we assess the power of the engine, the fuel efficiency, the number of seats, etc. Each one of these dimensions is referred to as an *attribute* which can take on a value from finite or possibly infinite domain. We focus on finite domains in this work. An outcome consists of an instantiation of all of these attributes and the set of all outcomes is the Cartesian product of the attribute domains. The number of outcomes grows exponentially with the number of attributes. However, there often exists some structure in how preferences over attributes are expressed that allows for a more compact representation of preferences over outcomes (Fishburn, 1970; Keeney and Raiffa, 1976; French, 1986).

One such structural assumption is *preferential independence*. Let $X_1, X_2, \ldots, X_n$ be the set of attributes. Each attribute has a finite domain; for ease of notation we will use X_i interchangeably to refer to both the i^{th} attribute and its domain. Let $\mathcal{X} = X_1 \times X_2 \times \cdots \times X_n$ be the set of all outcomes, let $I \subseteq \{1, 2, \ldots, n\}$ be the indices of a subset of attributes, and let the compliment I^c denote $\{1, 2, \ldots, n\} \setminus I$. Then the subset of attributes X_I are preferentially independent of the attributes X_{I^c} when the following is true:

$$(x_I, y) \succeq (x'_I, y) \text{ for some } y \in X_{I^c} \Rightarrow (x_I, y') \succeq (x'_I, y') \text{ for all } y' \in X_{I^c}$$

where the symbol $\succeq$ indicates a weak preference ordering (Keeney and Raiffa, 1976).

This kind of independence leads to an intuitive expression of preferences using ceteris paribus (all else being equal) statements of the form $x_I \succeq x'_I$. This statement is interpreted as $(X_I, y) \succeq (X'_I, y) \;\; \forall \; y \in X_{I^c}$. An expression of preferences using ceteris paribus statements about subsets of attributes can be represented using a graphical model such as *CP-nets* (Boutilier, Brafman, Domshlak, Hoos, and Poole, 2004a). Like graphical models for probability distributions, an attribute X_i is connected to other attributes that influence preferences over X_i. The CP-net can be used to compare full outcomes, and find the most preferred outcome in polynomial time (in the size of the CP-net); however, in general CP-nets cannot express arbitrary full preference orderings.

In a similar manner to preferences, the quantitative representation of utilities can be greatly reduced by introducing independence assumptions (Fishburn, 1970; Keeney and Raiffa, 1976). A common approach defines *additive* utility in terms of each attribute in isolation and assumes that the user is indifferent among lotteries that have same marginals on each attribute. Given this independence assumption, one can define local, attribute-specific utility functions such that the utility of an outcome is equal to the sum of the utility of each attribute which in turn can be represented by a scaled local value function:

$$u(\mathbf{x}) = \sum_{i}^{n} u_i(x_i) = \sum_{i}^{n} \lambda_i \upsilon_i(x_i) \tag{2.2}$$

The local value function υ_i is defined over the single attribute X_i and the scaling factor λ_i calibrates the local value function, expressing how much it impacts the global utility function. Let $x_i^\top$ be the most preferred level and $x_i^\perp$ be the least preferred level of attribute X_i. Then the local value function can be specified relative to these "anchor" levels, and the calibration weights λ_i set by finding the true utility of the "anchor" levels relative to some full outcome x Fishburn (1970).

When there is some dependence among attributes, *generalized additive independence* (GAI) models can be used to capture preference structure and decompose the value function into local utility functions over set of attributes (Fishburn, 1970):

$$u(\mathbf{x}) = u_1(x_{I_1}) + u_2(x_{I_2}) + \cdots + u_n(x_{I_n})$$

where the subsets I_i of attributes are mutually independent. For instance, in a four attribute domain, given $I_1 = \{1, 2\}, I_2 = \{3, 4\}, I_3 = \{2, 3\}$, then

$$u(x_1, x_2, x_3, x_4) = u_1(x_1, x_2) + u_2(x_3, x_4) + u_3(x_2, x_3).$$

A similar, but more sophisticated technique can be used to specify the utility for the GAI model in terms of local value functions (Fishburn, 1970). However, even when utility is decomposed into a set of value functions, each with respect to a single set of attributes, we cannot expect a user to be able to directly specify a numerical value for each $v_i(x_i)$ in an accurate and consistent way (French, 1986). The next section examines alternatives to directly specifying a numerical value.

2.1.4 Query Types

Queries can be used to directly or indirectly elicit information about a user's utility function. The most desirable queries are those that yield a high amount of information, while not putting a high cognitive load on the user. Queries can elicit *global information* with respect to the utility of whole outcomes, or can elicit *local information* about value functions over subsets of attributes.

Comparison A *global comparison query* asks a user to directly compare two of outcomes (or lotteries). For example: is outcome x preferred to outcome y? In the case of outcomes with few attributes this query often requires little cognitive effort on behalf of the user, but it also elicits little information, providing a simple linear constraint on utility: if $x \succeq y$ then $u(x) \geq u(y)$. In a multi-attribute model, a *local comparison query* asks a user to compare two instantiations of a subset of attributes, assuming that (ceteris paribus) the unspecified attributed are fixed.

Most Preferred A *global choice set query* asks the user which in a set of n outcomes is most preferred. This places a higher cognitive load on the user, but yields information equivalent to $n-1$ comparison queries (French, 1986) and places $n-1$ linear constraints on the utility function.

Equivalence A *global equivalence query* asks the user to choose a probability p for which the user indifferent between the lottery $p, \langle x^\top, (1-p), x^\perp \rangle$ and the certain outcome y. This is known as a *standard gamble query* and given that we know the utility of $x^\top$ and $x^\perp$ will directly specify the utility of y. It is unlikely that the typical user will be able to easily answer this type of query with the required level of precision. A generalization of the equivalence query is the *bound query*, which asks the user to directly bound the utility of an outcome or attribute from above or below.

Ranking A *ranking query* asks the user to rank a subset (or the entire set) of outcomes. The query is cognitively demanding since it requires preference information relating every pair of alternatives. While this query specifies a total ordering over the subset, it does not reveal the strength of preference between two outcomes.

Information about the preferences of a user can also be gathered implicitly by passively observing the choices of the user. Section 2.1.6 discusses conjoint analysis and collaborative filtering techniques which use aggregate information about the preferences of many users to infer the preferences of a separate individual user. Section 2.3.4 discusses a method for deriving the equivalent of utility functions for multi-step decision problems using a notion of observed behaviour.

In practice it can be extremely time consuming and cognitively demanding to entirely specify a utility function with full precision. The next section outlines criteria that have been proposed for making decisions on the basis of partially specified utilities.

2.1.5 Criteria for Decision Making with Partial Preferences

The partial information regarding preferences is usually assumed to take one of two possible forms. The first form assumes that the parameters of the feasible utility functions lie in some bounded region. These bounds can be derived using many of the queries from Section 2.1.4. This is referred to as the *strict uncertainty setting*. The second form of uncertainty assumes that we can characterize what we know in terms of a probability distribution over the parameters of the unknown utility function. Such distributions can be derived by aggregating the elicited utility functions of other users or by a domain expert specifying degrees of belief in possible utility functions for a user.

2.1.5.1 Strict Uncertainty

Strict uncertainty over preferences is characterized by allowing the decision maker's actual utility function to be one from a set of feasible utility functions U. The goal is then to realize the "best" outcome given a set of feasible utility functions. However, there are many reasonable measures of what the "best" outcome might be:

Maximax The *maximax criterion* represents the most optimistic approach. The criterion chooses the outcome that yields the highest utility over all of the feasible utility functions. However, the "true" utility function may yield a far lower utility for the chosen outcome than predicted.

$$\text{[Maximax]} \qquad x^*_U = \operatorname*{argmax}_{x \in X} \max_{u \in U} u(x)$$

Maximin The *maximin criterion* chooses the optimal outcome with respect to the worst choice of utility function (Wald, 1950). This yields a guarantee that the realized utility will be no worse than the maximin utility. However, this is a pessimistic measure that may be far too conservative with its choice of outcome.

$$\text{[Maximin]} \qquad x^*_U = \operatorname*{argmax}_{x \in X} \min_{u \in U} u(x)$$

Optimism-Pessimism Index The maximax and maximin criteria can be balanced by choosing an outcome which maximizes a combination of the utility functions chosen by each (Wald, 1950). The *optimism-pessimism index* (OPI) α determines how skewed the result is towards being pessimistic or optimistic.

$$[\text{OPI}] \qquad x^*_U = \operatorname*{argmax}_{x \in X} \Big[\alpha \min_{u \in U} u(x) + (1-\alpha) \max_{u \in U} u(x) \Big]$$

However, it is unclear how to choose the parameter α in a principled way.

Minimax Regret The *minimax regret* (MMR) criterion (Savage, 1951) compares the outcomes for each state of uncertainty. The maximum regret of choosing x is defined as

$$MR(x, U) = \max_{u \in U} \max_{x' \in X} \Big[u(x') - u(x) \Big]$$

which can be thought of as measuring worst case loss with respect to possible realizations of the utility function $u \in U$. The minimax regret criterion minimizes this worst case loss.

$$[\text{MMR}] \qquad x^*_U = \operatorname*{argmin}_{x \in X} MR(x, U) = \operatorname*{argmin}_{x \in X} \max_{u \in U} \max_{x' \in X} \Big[u(x') - u(x) \Big] \tag{2.3}$$

The minimax regret criterion is less pessimistic than maximin while still providing a guarantee with respect to worst case loss. This loss is arguably a more intuitive measure for a user since it is gauges how much better the user could have done if they knew the true utility.

2.1.5.2 Bayesian Uncertainty

Bernoulli argued that in the absence of any information about the probability of events, we should assume each event is equally likely (Hacking, 1971). This has been referred to as the *principle of insufficient reason*. A criterion based on this should choose the outcome x that maximizes the mean w.r.t. the feasible utility functions as follows

$$x^* = \operatorname*{argmax}_{x \in X} \frac{1}{|U|} \sum_{u \in U} u(x) \tag{2.4}$$

We can recast the above formulation in Bayesian terms by finding the outcome which maximizes expected utility with respect to a distribution σ over possible utility functions.

$$x^* = \operatorname*{argmax}_{x \in X} \mathbb{E}^{\sigma}_{u \in U} \Big[u(x) \Big] \tag{2.5}$$

Note that this is equivalent to formulation (2.4) when utility functions are drawn from a uniform distribution. Formulation (2.5) is also known as the "Bayesian criterion for imprecise utility" and has been used by a number of researchers (Boutilier, 2002; Chajewska, 2002). An example of how to construct a prior over utility functions can be found in the work of Chajewska and Koller (2000). They use a database of previously elicited utility functions to estimate the distribution of utility functions in the population and then use this estimate as a prior for each individual user.

Another possible criterion used by Delage and Mannor (2007) attempts to balance the strengths of the Bayesian and strict uncertainty approaches. Delage and Mannor refer to this approach as *percentile optimization* (it is also known as chance-constrained optimization) and it is directly related to the Value-at-Risk measure for mitigating market risk (Jorion, 1997). The criterion makes use of prior information while offering a form of probabilistic guarantee. The percentile criterion for choosing an outcome given a set of feasible utility functions is as follows:

$$x^* = \operatorname*{argmax}_{x \in X} \max_{y} \Pr\Big(u(x) \geq y \Big) \geq \eta$$

where η can be thought of as a confidence interval. Thus, when $\eta = 0.95$ we can think of the percentile criterion as finding the outcome x^* that yields the highest expected value, y, 95% of the time. Unfortunately this approach doesn't entirely realize the strength of either the Bayesian or strict uncertainty approaches, since it is not truly Bayesian and cannot offer a real (non-probabilistic) performance guarantee.

2.1.5.3 Query Selection

Given a parameterized partial preference representation and a type of query, the next step is to select which parameters to query in order to improve the quality of the decision according to the chosen criterion.

Under the assumption of strict uncertainty, the space of feasible parameters of the unknown utility function are often restricted to form a convex polytope. Some examples of how the initial polytope can be specified are 1) placing upper and lower bounds on each utility parameter or 2) bounding each utility parameter using a set of linear constraints imposed by an initial set of bound or comparison queries.

All responses to the queries in Section 2.1.5.1 impose either a linear bound or a linear inequality constraint on the utility polytope that potentially reduce its volume. With this geometric interpretation in mind, there are number of strategies that seek to significantly reduce volume using as few queries as possible.

The Q-Eval approach developed by Iyengar (2001) uses a comparison query whose response imposes a hyperplane which partitions the utility polytope into two components. A response to the query essentially eliminates one component. Iyengar uses a heuristic to choose the outcomes (and the resulting hyperplane) that will form the next query. The heuristic uses the distance of the hyperplane from the analytic center of the polytope to prune potential queries. The volume of the partitions resulting from each hyperplane are examined to find the query/hyperplane that will make these volumes as close to equal as possible.

The polyhedral methods of Toubia (2003b; 2004) use queries which ask the user to choose the most preferred of a set of n outcomes. A response yields $n - 1$ pairwise preference constraints and the hyperplanes imposed by each potential response create an n-partition of the polytope. Query selection proceeds by attempting to find the set of outcomes that equally partitions polytope. This problem is approximately solved by first finding the analytic center of the polytope and then using an ellipsoid to approximate the volume of the polytope.

One benefit of these volumetric approaches is that they yield bounds on the number of queries required to reduce to volume to some level ϵ. However, the end goal is to improve the quality of decision with respect to one of the criteria described in Section 2.1.5.1 and a reduction in the volume of the utility polytope does not guarantee an improvement w.r.t. the decision criteria. Boutilier et al. (2005) describe heuristic strategies for directly improving the minimax regret criterion. Given an uncertain factored utility function, their strategies select bound queries that ask about the midpoint of the interval constructed around the upper and lower bounds at a point in the utility polytope. The *halve the largest gap* strategy selects a point with the largest distance between its upper and lower bound. While the *current solution* strategy uses information from the solution x and x' of the minimax regret calculation (Equation (2.3)) in conjunction with the size of the interval to select the point at which to query. In practice these heuristic strategies have been shown to often outperform volumetric approaches.

Let $\mathcal{P}$ be the set of potential query responses. A Bayesian approach to selecting a query assumes that a prior ϕ can be established over possible utility functions, and further, that we can establish a distribution $\Pr(\rho \mid q, u)$ over responses $\rho \in \mathcal{P}$ to a query $q \in Q$ given a utility function u (this distribution can also incorporate a noise model allowing users to occasionally give an "incorrect" response). The approach used by Chajewska et al. (2000) is then to select the query that maximizes the *myopic expected value of information* (myopic EVOI).[1] To calculate myopic EVOI the posterior distribution $\Pr(u'|q, \phi)$ is found for each possible query. This is used to find the expected posterior utility EPU for each query, the EPU is then compared to the maximum expected utility (MEU) given the prior distribution over possible utilities and

[1]The expected value of information is myopic because it only considers the responses to next query and not how the next query will effect the responses of future queries.

the query is selected that yields the most gain:

$$\max_{q} EVOI(q, \phi) = EPU(q, \phi) - MEU(\phi)$$

One challenge of this approach is that the computational cost of updating the posterior $\Pr(u'|q, \phi)$ after each query response can be very high.

The myopic EVOI approach can be improved upon by computing the optimal *sequence* of queries. Boutilier (2002) and Holloway et al. (2003) have developed models for computing optimal sequential query policies offline using a partially observable Markov decision process. However, computing the optimal sequential query policy with these models is intractable for all but unrealistically small problems.

2.1.6 Preference Elicitation in Practice

Some of the earlier work that incorporates a computational approach to preference elicitation is that of White (2003) on multi-objective computer aided design. User preferences are used to guide the exploration of a large and complex design space by a variety of search strategies. Partial preferences are characterized by *imprecisely specified multi-attribute utility theory* (ISMAUT) in which the utility function is represented by a normalized sum of attribute-specific value functions. Approaches using ISMAUT have been applied to many problems from the designing communication networks (Edward A. Sykes, 1985) to fossil fuel boilers (Brown and White, 1987). The goal is to incorporate constraints on the value functions and generate a set of non-dominated options that a user may choose among. One potential issue with this approach is that the set of non-dominated options can be too large for a human user to reasonably assess. In this case, elicitation approaches have been proposed to narrow the set of non-dominated options, however, these approaches lack a principled query selection process.

The field of conjoint analysis grew out of research in consumer marketing (Green and Rao, 1971), but shares many similarities with work in preference elicitation. The goal of conjoint analysis is to take the aggregate preferences of consumers for products and decompose them to yield preferences over more general product attributes. Consumer preferences over these general attributes are then used to predict the success of future products. Like the work with IS-MAUT, conjoint analysis often assumes an additive utility function. The prediction of attribute preferences is accomplished using some form of regression, such as least squares (Green and Srinivasan, 1978, 1990). While conjoint analysis is a useful tool for constructing models of preference for groups of users, it does not address how preferences can be constructed for a single user.

Since the work of Konstan et al. on GroupLens (Konstan, Miller, Maltz, Herlocker, Gordon, and Riedl, 1997), collaborative filtering has been a popular approach for predicting the preferences of users. Much like conjoint analysis, collaborative filtering builds these predictions using the preferences of a large group of related users. This prediction has been constructed in many ways using item-to-item similarity (Sarwar, Karypis, Konstan, and Reidl, 2001), user-to-user similarity (Kautz, Selman, and Shah, 1997) and more recently matrix factorization (Rennie and Srebro, 2005) which learns an implicit set of features for each item and for each user that matches the observed preferences of users. The implicit features are then used to predict the unobserved preferences of users. Much of the work on collaborative filtering solves a single prediction task without actively eliciting further preferences through queries. However, collaborative filtering methods can be used to iteratively (Boutilier, Zemel, and Marlin, 2003b) query users in order to improve their predictions. Like conjoint analysis, collaborative filtering embodies a set of assumptions that are different from classical preference elicitation. It assumes the presence of a large (but sparse) set of ratings by other users when evaluating the preferences of an individual user.

The work of Pu and Faltings (2003) on *user-involved preference elicitation* builds on research in behavioural decision theory and incorporates affordances that aim to mitigate issues with how people have been observed to make decisions. This work advocates a flexible ordering in the elicitation process, allowing the user to choose the focus of the elicitation at each step. Examples of this flexibility are present in the work of Viappiani et al. (2006a; 2006b) and Reilly et al. (2004; 2005; 2007) on example critiquing. The process of example critiquing presents the user with an example and allows the user to place constraints on one of many attributes of the example, for instance requiring a car to cost less than \$15,000 dollars. The process iterates as the system generates a new example based on the refined user preferences. Example critiquing systems have been used to elicit user preferences for a variety of consumer choices, from rental apartments to travel plans (Faltings, Pearl, and Torrens, 2004). While these approaches offer a variety of novel interaction paradigms, they often lack a principled and precise model of the user's utility function.

There are many applications of preference elicitation to single-stage decision problems, but few that consider an agent making multiple decisions over time. In the next section we will describe the standard representation for decision problems that involve time and uncertainty.

2.2 Sequential Decision Making

This section shifts focus to the fundamentals of sequential decision making. We begin with a review of Markov decision processes (MDPs) and algorithms for computing optimal policies with emphasis on the linear programming approaches that will form the basis of exact algorithms for computing minimax regret.

2.2.1 Model

The *Markov decision process* (MDP) is a powerful formalism for representing an agent making a sequence of decisions in an uncertain environment. Much of the theory and notation for the discounted MDPs we will discuss was first developed by Howard (1960). A comprehensive treatment can be found in the text by Puterman (1994). In the MDP formalism, given the current state of the world, the agent chooses an action. This action has some stochastic effect which changes the state. The user then receives some reward based on the current state and chooses the next action, repeating the process.

Formally a (finite) Markov decision process is described by the tuple $\langle S, A, P, r, \beta \rangle$ where S is a finite set of states, A is a finite set of actions, the transition function $P : S \times A \rightarrow \Pi(S)$ maps the state and action to distribution over next states, the reward function $r : S \times A \rightarrow \mathbb{R}$ defines the reward received after taking an action in a specific state, and the starting state distribution β expresses the probability $\beta(s)$ that the process will begin in state s. In this model the state at time $t + 1$ depends only on the state at time t and the action taken. This is referred to as the *Markov property* and allows for efficient algorithms that decompose the problem by time step.

We describe an MDP as having *finite horizon* when the process terminates after a fixed number of time steps. When there is no stopping point the MDP is described as having an *infinite horizon*. In the finite horizon case, the goal is to maximize the total expected sum of rewards $E[\sum_{t=0}^{k-1} r_t]$ over k time steps. When there is an infinite horizon, a *discount factor* $\gamma \in [0, 1)$ is often introduced to favour immediate reward. The goal is then to maximize the total expected sum of discounted rewards $E[\sum_{t=0}^{\infty} \gamma^t r_t]$. The discount factor also has the effect of bounding the infinite sum to yield a finite amount of total reward.

A policy prescribes an action[1] to the agent for each possible state of the decision process. When the agent's behaviour is not dependent on the time step, a *stationary policy* $\pi : S \rightarrow A$ chooses an action given the current state. In an MDP with a finite horizon, a *non-stationary policy* $\{\pi_1, \pi_2, \ldots, \pi_T\}$ chooses an action for each state and time step $1, \ldots, T$. Given an infinite horizon, Howard (1960) shows that there must exist an optimal stationary policy, since there is always an infinite amount of time remaining.

Given an MDP and a stationary policy π, the total expected value of executing the policy with t steps left in a finite horizon is modeled by the *value function* V_t^π, recursively defined as

[1]Stochastic policies offer a distribution over actions for each state.

follows:

$$V_t^\pi(s) = r(s, \pi_t(s)) + \gamma \sum_{s' \in S} P(s, \pi_t(s), s') V_{t-1}^\pi(s') \tag{2.6}$$

$$V_1^\pi(s) = r(s, \pi_1(s)) \tag{2.7}$$

In the infinite horizon case the value of a policy V_π is independent of the time step and can be expressed as a fixed point of the following equation.

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum^{s' \in S} P(s, \pi(s), s') V^\pi(s') \tag{2.8}$$

This recursive formulation is the basis for the following dynamic programming algorithms that compute the optimal policy.

2.2.2 Computing Optimal Policies

Let V^* be the *optimal value function* w.r.t. to possible policies. Bellman (1957) established the following relationship between V^* at stage t and $t-1$.

$$V_t^*(s) = \max_a \; r(s, a) + \gamma \sum_{s' \in S} P(s, a, s') V_{t-1}^*(s') \tag{2.9}$$

This relationship is the basis of the *value iteration algorithm* which computes the optimal policy by first finding the value of the action that yields the highest reward with $t = 0$ steps to go and working backward. Algorithm 1 gives the details of the value iteration algorithm for a finite horizon T. At each step a policy is computed that maximizes the Q-value function $Q(s, a)$ which captures the value (given the current value function) of taking the immediate action a in state s and following the optimal policy thereafter.

For MDPs with infinite horizons (with a discount factor less than one) we can apply a similar approach, iteratively computing the value of the optimal policy at each time step. In

Algorithm 1: Value Iteration for Finite-Horizon MDP

```
$V_0(s) = \max_a r(s,a)$
$\pi_0(s) = \mathrm{argmax}_a r(s,a)$
**foreach** $t = 1 \dots T$ **do**
    $Q_t(s,a) = r(s,a) + \sum_{s \in S} P(s,a,s')V_{t-1}(s')$
    $V_t(s) = \max_a Q_t(s,a)$
    $\pi_t(s) = \mathrm{argmax}_a Q_t(s,a)$
**end**
```

this case the sequence of value functions V_t will linearly converge to the value of the optimal policy V_{π^*}. Let $\mathbf{V}$ denote the vectorized value function with entries V(s) and let the error ϵ at step t of value iteration be defined by the uniform norm $|| \bullet ||_\infty$. A policy that is within ϵ of optimality (i.e., *ϵ-optimal*) is guaranteed after the following stopping criterion has been met:

$$||\mathbf{V}_{t+1} - \mathbf{V}_t||_\infty < \frac{\epsilon(1-\gamma)}{2\gamma}$$

An alternative approach called *policy iteration* finds the policy directly (Howard, 1960). It comprises two steps: the policy evaluation step in which the value of the current policy is computed, and the policy improvement step which performs one backup and recomputes the best policy. Algorithm 2 gives the details of the policy iteration algorithm. When no improving action is available for any state, policy iteration has converged and the resulting policy is provably optimal (Puterman, 1994).

Value iteration has been show to converge linearly at a rate that is equal to the discount factor γ (Puterman, 1994). Due to the fact that policy iteration is directly modifying the policy at each iteration, it has been shown under some conditions to converge quadratically (Puterman, 1994). Many variants to policy iteration have been developed such as Modified Policy Iteration (Puterman and Shin, 1978) and Asynchronous Value Iteration (Gullapalli and Barto, 1994) which improve on convergence time.

The optimal value function can also be found using a single linear program (LP) that di-

Algorithm 2: Policy Iteration for Infinite-Horizon MDP

```
Initialize $\pi(s)$ to a random policy
while $V^{\pi_i}$ has not converged do
    Compute $V_\pi$ based on current policy $\pi$ [Policy Evaluation]
    foreach $s \in S$ do
        Find $a^*$ that maximizes $Q(s,a) = r(s,a) + \gamma \sum_{s \in S} P(s,a,s')V_\pi(s')$
        If $Q(s,a^*) > V^\pi(s)$ then $\pi(s) = a^*$, else $\pi(s) = \pi(s)$    [Policy Improvement]
    end
end
```

rectly solves the system of linear equations which encode the value function (Puterman, 1994).

$$\begin{aligned} \underset{V}{\text{minimize}} \quad & \sum_{s \in S} \beta(s) V(s) \\ \text{subject to} \quad & V(s) - \gamma \sum_{s' \in S} P(s,a,s') \geq r(s,a) \qquad \forall\, s,a \end{aligned} \tag{2.10}$$

This formulation allows the optimization to be performed by one of the many fast LP solvers currently available. The linear program has $|S|$ variables and $|S||A|$ constraints allowing for small to medium sized MDPs to be solved efficiently.

2.2.3 Occupancy Frequencies

While following a policy π, the *occupancy frequency* $f(s,a)$ corresponds to the discounted probability of being in a state s and taking action a. The set of feasible occupancy frequencies $\mathcal{F}$ for an MDP is defined with respect to an initial starting state distribution β:

$$\mathcal{F} \equiv \left\{ f \;\middle|\; \sum_{s_0} f(s_0) - \gamma \sum_{s,a} \Pr(s_0 \mid s,a) f(s,a) = \beta(s_0) \quad \forall\, s_0 \in S \right\} \tag{2.11}$$

The occupancy frequencies corresponding to the optimal policy are found with the following LP (Puterman, 1994):

$$\begin{aligned}
\underset{f}{\text{maximize}} \quad & \sum_{s\in S}\sum_{a\in A} f(s,a)r(s,a) && && (2.12)\\
\text{subject to} \quad & \sum_{a} f(s_0,a) - \gamma \sum_{s,a} \Pr(s_0 \mid s,a) f(s,a) = \beta(s_0) && \forall\, s_0 \in S\\
& f(s,a) \geq 0 && \forall\, s \in S, a \in A
\end{aligned}$$

This LP can be derived as the dual of LP (2.10) that explicitly encodes the value function. In general occupancy frequency correspond to a probabilistic policy π_f:

$$\pi_f(s,a) = \frac{f(s,a)}{\sum_{a'} f(s,a')}$$

When the occupancy frequencies are optimal with respect to a given reward function r, i.e., when

$$f^* = \underset{f\in\mathcal{F}}{\operatorname{argmax}} \sum_{s\in S}\sum_{a\in A} f(s,a)r(s,a), \tag{2.13}$$

then there is an optimal, deterministic policy π^* w.r.t. to r such that (Puterman, 1994):

$$\sum_{s\in S}\sum_{a\in A} f(s,a)r(s,a) = \sum_{s\in S} \beta(s)V(s) \tag{2.14}$$

In what follows we occasionally refer to occupancy frequencies and policies interchangeable since one uniquely determines the other.

2.2.4 Vector Notation

The proceeding vector notation allows for more concise expressions of the formulae and mathematical programs to follow.

Let $\mathbf{r}$ be an $|S| \times |A|$ matrix with entries $r(s, a)$. Let $\mathbf{P}$ be an $|S||A| \times |S|$ matrix. Restrictions of each matrix to action a are denoted $\mathbf{r}_a$ and $\mathbf{P}_a$ respectively. For notational convenience we define the matrix $\mathbf{E}$ to be identical to $\mathbf{P}$ with 1 subtracted for each self-transition probability $P(s|s, a)$. Let $\mathbf{f}$ denote a $|S||A|$ vector with entries $f(s, a)$ and let $\boldsymbol{\beta}$ denote an $|S|$ length vector with entries $\beta(s)$. Thus we can succinctly express the occupancy frequency constraints from Equation (2.11) as follows:

$$\mathcal{F} \equiv \left\{ \mathbf{f} \;\middle|\; \gamma \mathbf{E}^{\top} \mathbf{f} + \boldsymbol{\beta} = \mathbf{0},\ \mathbf{f} \geq \mathbf{0} \right\} \tag{2.15}$$

2.2.5 Scaling MDP Algorithms

A large amount of research has gone into representations and algorithms that allow optimal policies to be found for large MDPs. In a similar manner to the way outcomes are represented in multi-attribute utility functions, the structure of a state $\mathbf{x}$ can often be factored into a set of state attributes $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle$, each with a finite domain.[1] A dynamic Bayes-net can then be used to represent the transition function of the MDP and encode independence relationships between the state attributes (Boutilier, Dean, and Hanks, 1999). For example, the transition for a state attribute x_i is described by $\Pr(x_i^t \mid \text{parents}(x_i)^{t-1})$, where parents($x_i$) is the subset of attributes which influence the transition x_i. The full transition for an action a is then

$$\Pr(\mathbf{x}^t \mid \mathbf{x}^{t-1}, a) = \prod_i \Pr(x_i^t \mid \text{parents}(x_i)^{t-1}, a)$$

MDPs in which actions only affect a small set of state attributes can be represented far more compactly with a factored representation. Reward functions and actions spaces can be similarly factored and other structures such as *algebraic decision diagrams* (Hoey, St-Aubin, Hu, and Boutilier, 1999) have been used for even more concise representations.

Significant work has gone into *abstraction techniques* which solve the MDP by group-

[1]There are approaches for continuous domains, however, the focus of this section is on finite state MDPs

ing together states that are functionally equivalent (Boutilier, Dearden, and Goldszmidt, 1995; Boutilier and Dearden, 1994; Dean and Givan, 1997; Boutilier et al., 1999; Hoey et al., 1999). One possibility is grouping together and treating as one any states that have the same optimal action or have the same value in the value function. Abstraction can also group states that are not exactly the same to yield approximately optimal solutions introducing a tradeoff between the quality of the solution and the efficiency of the representation and policy calculation. Adaptive approaches aggregate states with similar value functions as the MDP is being solved. The reduction in the problem complexity afforded by these methods can yield a large speed-up in solution time. However, procedures for generating the abstraction often involve significant overhead and using these techniques on small MDPs with little regularity can in some cases be slower than standard approaches.

Decomposition techniques first separate the MDP into a set of smaller sub-MDPs which are solved independently in an efficient manner (Boutilier, Brafman, and Geib, 1997; Meuleau, Hauskrecht, Kim, Peshkin, Kaelbling, Dean, and Boutilier, 1998; Singh and Cohn, 1998). The locally optimal policies for the sub-MDPs are then combined to form an approximately optimal global policy. The sub-MDPs can be further decomposed to form hierarchies further exploiting any structure in the sub-MDPs (Dietterich, 2000; Andre and Russell, 2001). The intuition that drives these approaches is that MDPs can often be thought of as a set minimally interacting sub-processes. For instance, Meuleau et al. (1998) look at stochastic resource allocation problems in which there are natural sub-planning problems which are independent of each other given an allocation. Each sub-planning problem is formulated as an MDP and the value function is found for different resource allocation levels. A global policy is quickly constructed by using these value functions to compute a gradient for a heuristic search in the global MDP. When weakly interacting sub-processes are present in an MDP, decomposition allows for the solution of extremely large MDPs with more than 2^{1000} states. However, there is no guarantee that a given MDP will have such structure.

To further scale up MDPs computational techniques have been explored for approximating the value function using a linear combination of basis functions b_i: $\tilde{V}(\mathbf{x}) = \sum_i w_i b_i(x_i)$. Approximate value iteration, policy iteration and linear programming procedures have been developed which compute the weights w_i to minimize the error between the approximate value function and the true value function (Schuurmans and Patrascu, 2001; Poupart, Boutilier, Patrascu, and Schuurmans, 2002; Guestrin, Koller, Parr, and Venkataraman, 2003b). For problems in which the value function does not exhibit a naturally linear structure, neural networks have been applied (Zhang and Dietterich, 1996; Tesauro, 2002). When the number of basis functions (or nodes in the neural network) are small, techniques greatly reduce the complexity of storing and working with the value function, allowing for optimal policies to be found for MDPs with as many as 10^{40} states (Poupart et al., 2002).

Factoring, abstraction, decomposition and approximations of the value function allow for good policies to be found for extremely large MDPs; however, the rewards and transition probabilities for these MDP models must still be completely specified for these techniques to work. As we have previously discussed, the specification of the parameters (rewards and transitions probabilities) is often a time consuming process. The next section will mirror Section 2.1.5 and discuss how good policies can be computed for MDPs in which the parameters have only been partially specified.

Incorporating structure allows for concise representation of decision problems, however, considerable effort is still required specify the necessary model parameters (i.e., the state and action space, the transition function, and the reward function). The next section examines approaches for relaxing this requirement through partial parameter specification.

2.3 Sequential Decision Making with Partial Information

There are a number of existing approaches to sequential decision making when aspects of the model are not fully specified. This includes related work on partially observable MDPs (Kaelbling, Littman, and Cassandra, 1998) and inverse reinforcement learning (Ng and Russell, 2000). We pay particular attention to work that computes robust policies given an imprecise model of dynamics or reward.

The preference elicitation literature generally focuses on single stage decision problems, however, there are a few sub-fields of MDP research that contain the seeds of a preference elicitation approach to multi-step decision problems. We will first review the approaches to calculating optimal MDP policies when there is only partial information about the model. We will then discuss notions of elicitation that are used in conjunction with each approach.

2.3.1 Reinforcement Learning

We begin our discussion of MDPs with partial model information, by first examining a field that assumes no model information. Reinforcement learning involves an agent that learns behavior through trial-and-error interactions with a dynamic environment. This can be modeled as a partially specified MDP where initially the agent has no knowledge of the transition function P or reward function r. The agent begins in state s and must choose an action $a \in A$. Upon taking this action, the agent observes the realized transition (drawn from the stochastic model) to a state s' and receives a reward $r(s, a)$. The goal is to learn a policy online that maximizes the expected reward $\sum_{t=0} E\left[r(s_t, a_t)\right]$. This is different from MDPs where the optimal policy is computed offline.

There are two flavours of algorithms for learning policies in reinforcement learning settings that differ in their representation of the environment. Model-based reinforcement learning techniques such as E^3 (Kearns and Singh, 2002) and R-max (Brafman and Tennenholtz, 2003) learn and maintain an explicit model of the transition function while model-free techniques such as TD Learning (Sutton, 1988) and Q-Learning (Watkins and Dayan, 1992) attempt to assign value to states and actions without an explicit model of the transition function. In all approaches to reinforcement learning the agent must balance *exploration* actions which learn more about the environment with *exploitation* actions which take what is known and aim to maximize reward.

The reinforcement learning paradigm is attractive because it does not assume any prior knowledge about the dynamics of the environment or the reward function. However, because the agent must spend time exploring the environment, reinforcement learning algorithms often exhibit very slow convergence to the optimal (value maximizing) policy.

2.3.2 Robust MDPs

In many situations, a model of the transition dynamics can be estimated from measurements. However, due to issues like noise in the measurements or limited sample size, the resulting transition model may not be entirely accurate. This can result in policy that is optimal with respect to the measured transition model performing poorly in the real world. Work on robust MDPs aims to mitigate this concern by first allowing transition functions to be partial specified a priori using strict uncertainty, and then computing policies that are robust to possible worst case transitions functions.

The work of Iyengar (2005) looks at infinite horizon MDPs and uses the maximin criterion to compute a robust policy. The uncertainty over the transition function is represented by a set $\mathcal{P}$ of feasible transition functions that forms a convex polytope.[1] In this context the maximin criterion can be thought of in terms of a game where the agent first chooses a policy and an adversary is then free to choose a transition function P from the set $\mathcal{P}$ so as to minimize the total discounted reward received by the agent. The goal is to choose the policy π^* that performs the best given the adversary's ability to set the transition function P.

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \min_{P \in \mathcal{P}} \quad E_x^{P,\pi} \left[\sum_t \gamma^t r(x) \right] \tag{2.16}$$

Iyengar is able to efficiently optimize Equation (2.16) and find the robust policy by decomposing the problem across time steps. He uses a variant of the value iteration algorithm which

[1] In practice such a polytope could be constructed by placing bounds on probabilities of each transition, while enforcing that each probability form a valid distribution.

computes the best action given the worst transition at each time step. This dynamic programming decomposition essentially computes a set of Q-values (one for each action) and chooses the action with the highest Q-value. This reduces the problem to computing the Q-values by finding the worst case transition function given a specific action:

$$Q_t(s,a) = \min_{P \in \mathcal{P}} \; r(s) + \gamma \sum_{s'} P(s'|s,a) V_{t-1}(s) \tag{2.17}$$

There are a variety of approaches to the above optimization that depend on the structure of the set of feasible transition functions. For example, given an initial estimate of the transition function q, the set of feasible transition functions can be defined as $\mathcal{P} \equiv \{ \; p \mid D(p||q) \leq \delta \; \}$, where $D(p||q)$ is the relative entropy:

$$D(p||q) \equiv \sum_i p(i) \log\left(\frac{p(i)}{q(i)}\right)$$

The optimization in Equation (2.17) can then be converted by duality to a problem of minimizing a single-variable, convex function and a bisection method is used, with complexity of $O(|S| \log(V_{max}/\epsilon))$, where $\epsilon > 0$ specifies the error, and V_{max} is the global upper bound on the value function.

Bagnell, Ng $\&$ Schneider (2003) adopt the same maximin formulation as Iyengar, but they offer a different proof of the fact that the global optimization (Equation (2.16)) can be decomposed into a maximin optimization at each time-step (Equation (2.17)). They contribute a novel variation in which a cost term C is added to the value function that captures the energy that has been injected into the system by perturbations of the transition function caused by the adversary:

$$Q_t(s,a) = \min_{P \in \mathcal{P}} \; r(s) + \gamma \sum_{s'} P(s'|s,a) V_{t-1}(s) + \lambda C(P, s', s)$$

This cost term has the effect of bounding the power of the adversary, and producing less pessimistic policies.

Nilim & Ghaoui (2005) also formulate the robust optimization problem using the maximin criterion and Equation (2.16). However, they offer a richer characterization of the feasible transition functions. In addition to relative entropy, they allow for the set of transition functions to be bounded by likelihood, and maximum-a-posteriori (MAP) estimators. For bounded likelihood, they define feasible transitions $\mathcal{P}$ to be the likelihood region $\{\ p \mid \sum_i q(i) \log p(i) \geq \delta\ \}$ with some constant δ. In some settings there is a natural prior distribution that can be placed on the set feasible transition functions and this prior can be incorporated using a MAP estimator. Nilim & Ghaoui use the log MAP estimator $L_{map} = L(p) + g_0(p)$ where g_0 refers to the prior density function over p and $L(p)$ refers to the log-likelihood of p. The set of feasible transitions is then defined as $\{\ p \mid L_{map}(p) \geq \delta\ \}$. Both likelihood and MAP models result in a convex set of feasible transition functions and offer an efficient solution to the problem of finding the one-step worst case transition given an action (Equation (2.17)).

McMahan, Gordon & Blum (2003) show how to solve a similar problem, however, they find a maximin policy with respect to uncertain reward functions:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \min_{r \in \mathcal{R}} \ E_x^\pi \left[\sum_t \gamma^t r(x) \right]$$

In this setting the adversary chooses a reward function from a set of feasible reward functions $\mathcal{R}$ so as to minimize the total discounted reward. Rather than using a dynamic programming approach, the authors formulate the maximin optimization as a linear program using Benders' decomposition and use constraint generation to efficiently search through the space of all possible constraints in the following formulation:

$$\begin{aligned} &\max_{\delta,\pi} && \delta \\ &\text{subject to:} && \delta \geq V_r^\pi \qquad \forall\, r \in \mathcal{R} \end{aligned} \tag{2.18}$$

where V_r^π is the optimal value function with respect to the reward function r. This optimization is tractable for reasonably sized MDPs, however, the maximin criterion leads to conservative policies since it attempts to find the best policy for the worst case settings of reward. For instance, a policy that provides very high value for 99% of reward functions, but a very low value for 1% of reward functions would be rejected in favour of a policy that provides moderate value for all reward functions.

Delage & Mannor (2007) take a different approach and adopt a percentile-based criterion to find policies given partial information about transition and reward functions. The formulation is similar to that of Equation (2.5) in Section 2.1.5.2 as can be expressed as follows:

$$\begin{array}{ll} \max\limits_{y,\pi} & y \\ \text{subj:} & \Pr\Big(E(\sum\limits_{t=0}^{\infty} \gamma^t r_t(x_t) \mid \pi) \geq y\Big) \geq \eta \end{array}$$

Given some η, the formulation finds a policy that will performs at least as well as y in η percent of instances, where an instance is a setting of the uncertain parameters. They show how to approximately solve this formulation when the prior on the uncertain transition function takes form of a Dirichlet distribution. An algorithm for solving this formulation exactly for an uncertain reward function in the form of Gaussian is also described.

Each of the above formulations use criteria that assume strict uncertainty, with the exception of the percentile criterion of Delage & Mannor which takes a quasi-Bayesian approach given priors over reward and transitions. The next section will describe how partially specified model information can be characterized in a fully Bayesian form.

2.3.3 Partially Observable Markov Decision Processes

MDPs have been extended to model environments in which an agent has uncertainty over the current state. An example of such a setting is a mobile robot tracking its location with noisy sensors. *Partially Observable Markov Decision Processes* (POMDPs) assume that the current

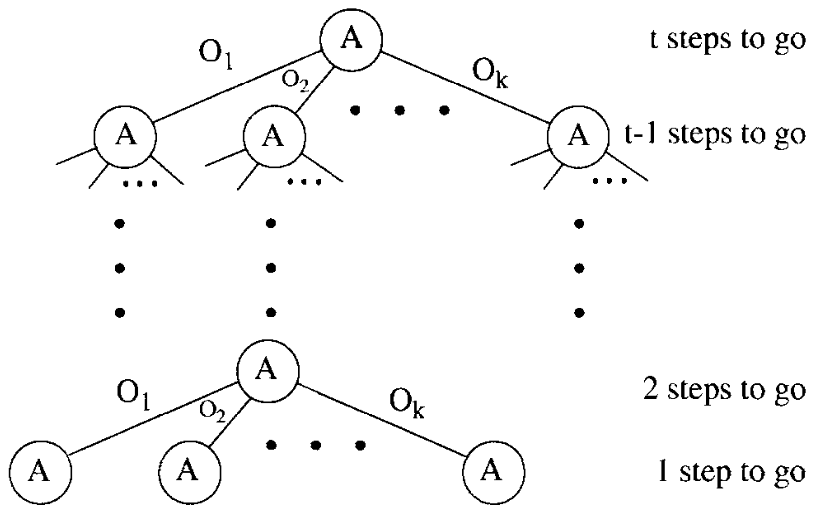


Figure 2.1: A t-step policy policy tree for a partially observable Markov decision process (Cassandra et al., 1994).

state is hidden; rather than directly observing the current state, the agent maintains a *belief state* b capturing distribution over possible states. Added to the model is a set of observations Ω that result from an action, and an observation function $\mathcal{O}(s, a, o)$ which encodes the probability of receiving an observation $o \in \Omega$ given an action a and a resulting state s'. The observation function is used to find the agent's updated belief b' given the previous belief b, and an action a:

$$b'(s') = \frac{\Pr(o|s', a, b)\Pr(s'|a, b)}{\Pr(o|a, b)} = \frac{\mathcal{O}(s', a, o)\sum_s P(s, a, s')b(s)}{\Pr(o|a, b)} \tag{2.19}$$

An agent's policy can be represented as a *policy tree* τ (shown in Figure 2.1) that specifies an action given a sequence of observations. Let a_τ be the action specified by the top node of the policy tree τ and let $o(\tau)$ be the $(t-1)$-step policy tree associated with observation o at the top level of a t-step policy tree τ. Given a starting state s, the value of a policy tree τ can be expressed as:

$$V_\tau(s) = r(s, a_\tau) + \gamma \sum_{s' \in S} P(s'|s, a_\tau) \sum_{o \in \Omega} \mathcal{O}(s', a_\tau, o) V_{o(\tau)}(s') \tag{2.20}$$

However, since states cannot be directly observed, the quantity of interest is the value $V_\tau(b)$ of the agent's current belief b, which is the expectation $\sum_{s \in S} b(s)V_\tau(s)$. Accordingly, a POMDP policy can be encoded as, $\pi_\tau(b) = a$, mapping a belief state b to action a.

The POMDP is an extremely general formalism. By augmenting state variables a POMDP can naturally represent any model parameters of an MDP that are unknown. The transition dynamics of the POMDP essentially model dynamics of the original MDP, while the observation dynamics and belief update is able to capture the uncertainty with respect to the model parameters. The benefit of this approach is that it allows for prior information over the uncertain model parameters to be incorporated in a manner that is principled and fully Bayesian. However, unlike robust approaches, it can only offer a probabilistic guarantee of performance in expectation. The augmented state space can be extremely large ($|S \times r|$ in the case of reward function uncertainty) and the problem of solving POMDPs can quickly become intractable as the state space grows (Poupart, 2000).

2.3.4 Inverse Reinforcement Learning

Another setting that addresses uncertainty over the reward function of the MDP is that of inverse reinforcement learning (IRL) (Ng and Russell, 2000). In this setting the partial information about the reward function is presented as observations of optimal behaviour, specifically a fixed set of observed sequences of actions and state transitions. The challenge is to recover the reward function and construct a generally optimal policy which is consistent with the observed behaviour. Given an observed policy π, Ng derives a set of constraints on a reward function that would induce π:

$$(\mathbf{P}_\pi - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1}\mathbf{r} \succeq \mathbf{0} \tag{2.21}$$

where $\mathbf{P}_\pi$ is the $|S| \times |S|$ matrix with entries $\mathbf{P}_\pi[s, s'] = P(s'|\pi(s), s)$. Recovering a reward function using simply the inverse reinforcement learning (IRL) constraints in Equation (2.21) is an ill-posed problem, since there are many possible solutions that make π optimal. For instance, the degenerate case where all rewards are zero is a solution. Different criteria have been proposed to find the "right" reward function. Along with their original formulation of the inverse reinforcement learning setting, Ng and Russell (2000) suggest some heuristics that aim to pick a reward function that maximally differentiates the observed policies from other sub-optimal policies. Ratliff et al. (2006) modify the problem to find a reward function that maximizes the margin in the IRL constraints. The work of Ramachandian et al. (2007) relaxes the hard constraints on reward and takes a Bayesian approach in which they consider the actions of the expert as evidence used to update a prior on reward functions. They pose the IRL problem as learning the posterior distribution on rewards. Ziebart et al. (2008) take a similar approach using a criterion based on maximizing entropy.

Inverse reinforcement learning approaches have been used for a number of tasks such as piloting a helicopter (Coates, Abbeel, and Ng, 2008) and finding optimal routes for taxis (Ziebart et al., 2008). However, they do not address how uncertainty can be further reduced through explicit interaction with a user who does not know an optimal policy a priori. In cases where an optimal policy can be demonstrated, inverse reinforcement learning can compliment the elicitation approach that we present by providing initial bounds on the reward function.

2.4 Example Application Domains

This section will introduce two application domains as background. These domains will be referenced throughout the thesis in examples and experiments to explain and assess our contributions to computing robust policies and eliciting reward. A third domain (website optimization) will be the subject of Chapter 7.

2.4.1 Autonomic Computing

The goal of autonomic computing is to build computer systems that self-manage obviating the need human intervention (Kephart and Chess, 2003). In large computing systems, such autonomy necessitates the continuous allocation and re-allocation of resources (e.g., units of storage or compute cycles) to distinct computing servers. When we incorporate the changing uncertain demands that are placed on individual servers, resource allocation in this setting is naturally modeled as a Markov decision process.

There is generally no closed-form for server reward; its precise specification, while automated through simulation, can involve significant expenditure of time and computation (Boutilier et al., 2003a). Reward specification in this domain can benefit from incremental elicitation. Queries bounding reward for specific demand/resource settings allow for provably optimal policies to be identified without a costly full specification of the reward function.

2.4.2 Assistive Technologies

This domain includes systems that provide cognitive assistance for persons with dementia, enabling them to complete the common activities of daily living. We focus on the COACH project (Boger, Poupart, Hoey, Boutilier, Fernie, and Mihailidis, 2005, 2006), whose goal is to guide a person through a small task (e.g., hand-washing) by providing verbal or visual cues, while allowing the individual to maintain as much independence as possible. Previous work in this domain specified reward through a time-consuming process of making small changes to a fully-specified reward function, assessing the resulting policy, and attempting to translate desired changes in system behaviour into further small changes to the reward function. Reward elicitation has the potential to dramatically streamline this process.

Looking Forward

The preceding sections of this chapter laid the groundwork for our reward elicitation framework while highlighting relevant related work. Looking forward, we focus on an extended Markov decision process that models partially specified reward functions using strict uncertainty. We adopt the minimax regret criterion and adapt existing approaches for computing minimax regret in single-step decision problems which we complement with methods inspired by the POMDP literature. We drive reward elicitation with decision-theoretic query selection approaches derived in part from single-step utility elicitation. We begin in the next chapter by laying our extended MDP model and detailing several approaches to minimax regret computation.

Chapter 3

Computing Robust Policies using Minimax Regret

We begin by formally defining the imprecise reward MDP (IRMDP) that is the focus of the thesis. Unlike the problem of finding an optimal policy given a known reward function, the problem of finding a minimax regret optimal policy for an IRMDP does not admit an obvious dynamic programming decomposition. Instead, we detail an exact method for computing minimax regret using constraint generation and mixed integer programming and demonstrate its effectiveness for small IRMDPs. We further define the set Γ of policies that are nondominated w.r.t. the imprecision in reward. Section 3.6 explores how this set of nondominated policies can be exploited to dramatically speed up minimax regret computation.

The work in this chapter primarily stems from the results of Regan and Boutilier (2008; 2009; 2010).

3.1 Imprecise Reward MDPs and Minimax Regret

We define the *imprecise reward MDP* (IRMDP) as the tuple $\langle S, A, P, \gamma, \beta, \mathcal{R} \rangle$ where the reward function r from a fully specified MDP is replaced by a *set* of feasible reward functions $\mathcal{R}$. The set $\mathcal{R}$ naturally arises from observations of user behaviour, partial elicitation of preferences, or information from domain experts, which typically place linear constraints on reward.

In this work we assume that $\mathcal{R}$ is a bounded, convex polytope defined by linear constraint set $\{r \mid \mathbf{C}\mathbf{r} \leq \mathbf{d}\}$. We use $|\mathcal{R}|$ to denote the number of constraints. These linear constraints naturally arise from user responses to a set of queries about the reward function that are detailed in Chapters 4 and 5.

To compute "good" policies given the set of feasible reward functions we enlist the minimax regret criterion, since it offers the tightest possible bound on reward loss of the policy over potential reward realizations. Let $\mathbf{f}, \mathbf{g} \in \mathcal{F}$ be occupancy frequencies (corresponding to MDP policies), and let $\mathbf{r}$ be a reward function. We define the following variants of regret:

$$\text{[Regret]} \qquad R(\mathbf{f}, \mathbf{r}) = \max_{\mathbf{g} \in \mathcal{F}} \; \mathbf{g} \cdot \mathbf{r} - \mathbf{f} \cdot \mathbf{r} \tag{3.1}$$

$$\text{[Pairwise Max Regret]} \qquad PMR(\mathbf{f}, \mathbf{g}, \mathcal{R}) = \max_{\mathbf{r} \in \mathcal{R}} \; \mathbf{g} \cdot \mathbf{r} - \mathbf{f} \cdot \mathbf{r} \tag{3.2}$$

$$\text{[Max Regret]} \qquad \begin{aligned} MR(\mathbf{f}, \mathcal{R}) &= \max_{\mathbf{r} \in \mathcal{R}} \; R(\mathbf{f}, \mathbf{r}) && (3.3) \\ &= \max_{\mathbf{g} \in \mathcal{F}} \; PMR(\mathbf{f}, \mathbf{g}, \mathcal{R}) && (3.4) \\ &= \max_{\mathbf{g} \in \mathcal{F}} \max_{\mathbf{r} \in \mathcal{R}} \; \mathbf{g} \cdot \mathbf{r} - \mathbf{f} \cdot \mathbf{r} && (3.5) \end{aligned}$$

$$\text{[Minimax Regret]} \qquad \begin{aligned} MMR(\mathcal{R}) &= \min_{\mathbf{f} \in \mathcal{F}} \; MR(\mathbf{f}, \mathcal{R}) && (3.6) \\ &= \min_{\mathbf{f} \in \mathcal{F}} \max_{\mathbf{g} \in \mathcal{F}} \max_{\mathbf{r} \in \mathcal{R}} \; \mathbf{g} \cdot \mathbf{r} - \mathbf{f} \cdot \mathbf{r} && (3.7) \end{aligned}$$

$R(\mathbf{f}, \mathbf{r})$ is the *regret* or loss of policy $\mathbf{f}$ relative to the fixed reward $\mathbf{r}$ and quantifies the difference in value between $\mathbf{f}$ and the optimal policy under $\mathbf{r}$. $MR(\mathbf{f}, \mathcal{R})$ is the *maximum regret* of the policy $\mathbf{f}$ w.r.t. the feasible reward set $\mathcal{R}$. Given a choice of policy $\mathbf{f}$, maximum regret represents the worst-case loss over possible realizations of reward, or as the regret incurred in the presence of an *adversary* who chooses the reward $\mathbf{r}$ to maximize this loss. Equivalently, max regret can be viewed as the adversary choosing a policy with greatest *pairwise max regret* $PMR(\mathbf{f}, \mathbf{g}, \mathcal{R})$, defined as the maximal difference in value between policies $\mathbf{f}$ and $\mathbf{g}$ under possible reward realizations.

In the presence of such an adversary, we wish to minimize the max regret. $MMR(\mathcal{R})$ is the *minimax regret* of feasible reward set $\mathcal{R}$. This can be seen as a game between a player choosing $\mathbf{f}$ to minimize loss relative to the optimal policy, and an adversary selecting the reward $\mathbf{r}$ to maximize this loss given the player's choice. We refer to any policy $\mathbf{f}^*$ that minimizes max regret as a *minimax optimal policy*. We denote the reward $\mathbf{r}$ that maximizes regret of $\mathbf{f}^*$ as the *adversarial reward*, and the optimal policy $\mathbf{g}$ for $\mathbf{r}$ as the *adversarial policy*. It is worth noting that minimax regret measures performance by assessing a policy *ex post* and making comparisons only w.r.t. specific reward realizations. Thus, policy $\mathbf{f}$ is penalized on reward $\mathbf{r}$ only if there exists a $\mathbf{f}'$ that has higher value w.r.t. $\mathbf{r}$ itself.

For our purposes, the minimax regret criterion has many desirable properties. Given a computed minimax regret optimal policy, the criterion gives an intuitive measure of the "goodness" of the policy by bounding the improvement in policy value that would be realized if one could eliminate all reward uncertainty. The criterion can also be viewed as measuring the impact of future elicitation by bounding the possible improvement in policy value. The solution to minimax regret also serves to inform elicitation, biasing query selection toward "high impact" queries. However, unlike related criteria (e.g., maximin) it does not admit a dynamic programming decomposition that allows for efficient computation. This is unsurprising given that the problem of finding the minimax regret optimal policy has been shown to be NP-hard in general (Xu and Mannor, 2009). Next we will detail a mathematical programming approach for exact minimax regret computation.

3.2 Randomly Generating IRMDPs

In addition to IRMDPs that model problems in the domains of autonomic computing, assistive technology, and website optimization, we use randomly generated IRMDPs to conduct a preliminary evaluation of computational approaches developed in this and future chapters.

We impose some structure on the randomly generated IRMDP by creating a semi-sparse transition function. For each (s, a)-pair, $\lceil \log n \rceil$ reachable states are drawn uniformly and a Gaussian is used to generate transition probabilities. We use a uniform initial state distribution β and discount factor $\gamma = 0.95$. An imprecise reward model is generated by:

1. The uniform selection of each true (but hidden) rewards $r(s, a)$ from a predefined range.
2. The random generation of an uncertain *interval* whose size is normally distributed.
3. The uniform random placement of the interval around the true $r(s, a)$.

The resulting set of feasible rewards forms a hyper-rectangle. We refer to this methodology as semi-sparse random IRMDP generation. Full details can be found in Appendix B.

It is not our goal to construct a set of IRMDPs that are accurate representations of real-world decision problems, but rather to provide a rough guide to selecting appropriate computational methods to then test on more realistic IRMDPs.

The parameters of the random generation process were chosen to find a middle ground where minimax regret computation is not unrealistically hard nor trivially easy. One measure of the complexity of the IRMDP w.r.t. to minimax regret computation is the number of nondominated policies that an IRMDP admits. In Section 3.6 we observe that the number of nondominated policies admitted by our random MDP generation procedure is far less than $|A|^{|S|}$, the total number of possible policies for an IRMDP with $|A|$ actions and $|S|$ states. At the same time, the number of nondominated policies admitted by a more realistic IRMDP—the website optimization setting explored in Chapter 7—is significantly less than the number of nondominated policies admitted by our randomly generated IRMDPs of similar dimension.

3.3 Computing Exact Minimax Regret

Following the formulations for non-sequential problems developed in (Boutilier et al., 2006; Boutilier, Sandholm, and Shields, 2004b), we formulate minimax regret optimization using a series of linear and mixed integer programs. We begin with formulation of minimax regret as the following mathematical program:

$$\begin{aligned} \underset{\mathbf{f}}{\text{minimize}} \quad & \max_{\mathbf{g}} \max_{\mathbf{r}} \quad \mathbf{g}\cdot\mathbf{r} - \mathbf{f}\cdot\mathbf{r} \\ \text{subject to:} \quad & \gamma\mathbf{E}^{\top}\mathbf{f} + \boldsymbol{\beta} = \mathbf{0}, \quad \mathbf{f} \geq \mathbf{0} && (3.8) \\ & \gamma\mathbf{E}^{\top}\mathbf{g} + \boldsymbol{\beta} = \mathbf{0}, \quad \mathbf{g} \geq \mathbf{0} && (3.9) \\ & \mathbf{Cr} \leq \mathbf{d} \end{aligned}$$

Here $\mathbf{f}$ is the minimax optimal policy, $\mathbf{g}$ is the adversary policy and $\mathbf{r}$ is the reward function. Constraints (3.8)–(3.9) are occupancy frequency constraints derived from Equation (2.11). We translate the minimax problem over the variables $\mathbf{f}, \mathbf{g}, \mathbf{r}$ into a minimization problem over $\mathbf{f}$ only by introducing an infinite (continuous) set of constraints, one for each $\langle \mathbf{g}, \mathbf{r} \rangle$ pair:

$$\begin{aligned} \underset{\mathbf{f},\delta}{\text{minimize}} \quad & \delta && (3.10) \\ \text{subject to:} \quad & \delta \geq \mathbf{g}\cdot\mathbf{r} - \mathbf{f}\cdot\mathbf{r} \qquad \forall\, \mathbf{g} \in \mathcal{F}, \mathbf{r} \in \mathcal{R} && (3.11) \\ & \gamma\mathbf{E}^{\top}\mathbf{f} + \boldsymbol{\beta} = \mathbf{0} \\ & \mathbf{f} \geq \mathbf{0} \end{aligned}$$

This roughly corresponds to the dual LP formulation of an MDP (introduced in Equation (2.12)) with the addition of adversarial constraints (3.11). These constraints enumerate the infinite set of feasible adversary policies and choices of reward. However, the number of active constraints at the optimal solution is often very small. Rather than minimizing w.r.t. to all possible constraints, we apply a common approach from the operations research literature, using

Benders' Decomposition (Benders, 1962; Nemhauser and Wolsey, 1988) to iteratively generate the set of active constraints at the optimal solution. Benders' Decomposition uses a convergent series of approximations to solve programs of the form in LP (3.10). At each iteration, two optimizations are solved. The *master problem* solves a relaxation of the original program using a subset of the adversarial constraints, corresponding to a subset *GEN* of all $\langle \mathbf{g}, \mathbf{r} \rangle$ pairs. We refer to the set *GEN* as the *generated constraints*. Intuitively, in the game against the adversary, this restricts the adversary to choosing a *witness pair* $\langle \mathbf{g}, \mathbf{r} \rangle$ from the set *GEN*.

$$\begin{aligned} MMR(\mathcal{R}, \mathit{GEN}) \;=\; & \underset{\mathbf{f},\delta}{\text{minimize}} \quad \delta && (3.12)\\ & \text{subject to:} \quad \delta \geq \mathbf{g}_i \cdot \mathbf{r}_i - \mathbf{f} \cdot \mathbf{r}_i \qquad \forall \langle \mathbf{g}_i, \mathbf{r}_i \rangle \in \mathit{GEN} && (3.13)\\ & \gamma \mathbf{E}^\top \mathbf{f} + \boldsymbol{\beta} = \mathbf{0} \\ & \mathbf{f} \geq \mathbf{0} \end{aligned}$$

Since we are restricting the adversary to choices in *GEN*, $MMR(\mathcal{R}, \mathit{GEN})$ forms a *lower bound* on the true minimax regret $MMR(\mathcal{R})$ (computed given an unrestricted adversary). The *subproblem* generates the maximally violated constraint relative to the current solution $\mathbf{f}$ to the master problem by removing restrictions on the adversary, computing maximum regret $MR(\mathbf{f}, \mathcal{R})$. We refer to the solution as the witness pair $\langle \mathbf{g}, \mathbf{r} \rangle$. $MR(\mathbf{f}, \mathcal{R})$ represents the regret associated with simply recommending policy $\mathbf{f}$ to the user and forms an *upper bound* on true minimax regret, If $MR(\mathbf{f}, \mathcal{R}) > MMR(\mathcal{R}, \mathit{GEN})$, then we add the new witness pair to *GEN* and continue the procedure (recomputing the master problem with the updated set of generated constraints). We terminate the procedure when the lower bound on minimax regret generated by the master problem is equal to the upper bound generated by the subproblem (i.e., $MMR(\mathcal{R}, GEN) = MR(\mathbf{f}, \mathcal{R})$).

The following mixed integer program formulates the subproblem of computing max regret using the MDP Value and Q-Value functions:[1]

$$\begin{aligned}
\underset{\mathbf{Q},\mathbf{V},\mathbf{I},\mathbf{r}}{\text{maximize}} \quad & \boldsymbol{\beta}\cdot\mathbf{V} - \mathbf{f}\cdot\mathbf{r} && && (3.14)\\
\text{subject to:} \quad & \mathbf{Q}_a = \mathbf{r}_a + \gamma\mathbf{P}_a\mathbf{V} && \forall\, a \in A && \\
& \mathbf{V} \geq \mathbf{Q}_a && \forall\, a \in A && (3.15)\\
& \mathbf{V} \leq (1 - \mathbf{I}_a)\mathbf{M}_a + \mathbf{Q}_a && \forall\, a \in A && (3.16)\\
& \mathbf{C}\mathbf{r} \leq \mathbf{d} && && \\
& \sum_a \mathbf{I}_a = \mathbf{1} && && (3.17)\\
& \mathbf{I}_a(s) \in \{0, 1\} && \forall a \in A, s \in S && (3.18)\\
& \mathbf{M}_a = \mathbf{M}^\top - \mathbf{M}_a^\perp && &&
\end{aligned}$$

The indicator vectors $\mathbf{I_a}$ represent the adversary's policy, with $I_a(s)$ denoting the probability of action a being taken at state s. The vector $\mathbf{M_a}$ serves to render constraint (3.16) non-binding when present (i.e., when some $I_a(s) = 0$). Constraints (3.17) and (3.18) restrict the policy to be deterministic and together with constraint (3.15) and (3.16) they ensure that the optimal value $V(s)$ will be set to the Q-Value $Q(s, a)$ for at a single action a—in fact $V(s)$ will be set to $\max_a Q(s, a)$. We ensure a tight $\mathbf{M_a}$ by setting $\mathbf{M}^\top$ to be the optimal value function $\mathbf{V}^\top$ of the optimal policy with respect to the best setting of each individual reward point and $\mathbf{M_a^\perp}$ to be the Q-value $\mathbf{Q_a^\perp}$ of the optimal policy with respect to the worst point-wise setting of rewards (the resulting rewards need not be feasible).

The subproblem formulation (3.14) does not directly produce a witness pair $\langle \mathbf{g}_i, \mathbf{r}_i \rangle$ for the master constraint set *GEN*. Rather it provides $\mathbf{r}_i$ and $\mathbf{V}_i$. We do not need access to $\mathbf{g}_i$ directly. The adversarial constraint (3.13) can be constructed from the reward function $\mathbf{r}_i$ and the value

[1] Specifying max regret in terms of occupancy frequencies (i.e., the standard dual MDP formulation (2.12)) gives rise to a non-convex quadratic program.

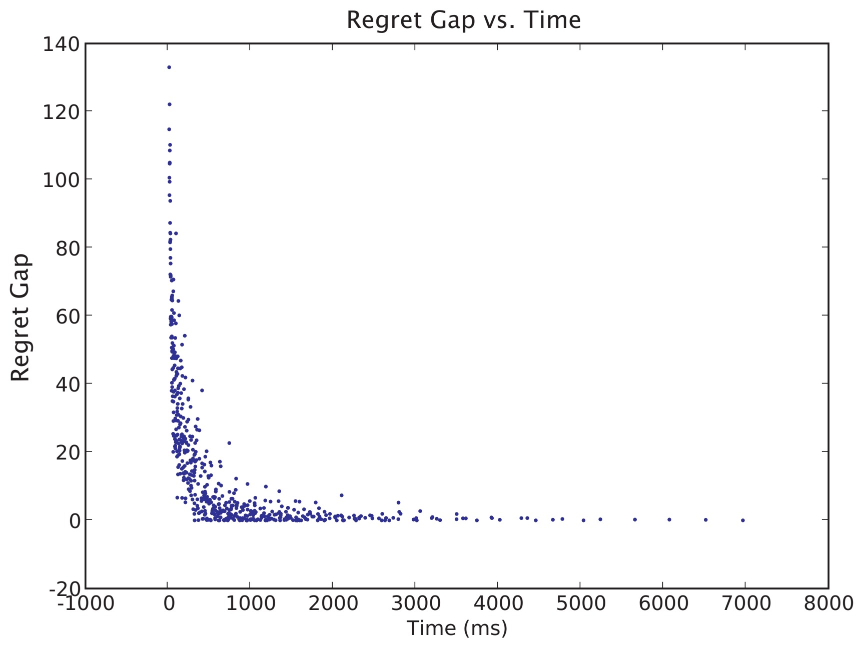


Figure 3.1: Reduction in regret gap during constraint generation

$\boldsymbol{\beta}\cdot\mathbf{V}_i$, since $\boldsymbol{\beta}\cdot\mathbf{V}_i = \mathbf{r}_i\cdot\mathbf{g}_i$. As a consequence, we use the following modified master formulation:

$$\begin{aligned} &\underset{\mathbf{f},\delta}{\text{minimize}} && \delta && && (3.19)\\ &\text{subject to:} && \delta \geq \boldsymbol{\beta}\cdot\mathbf{V}_i - \mathbf{f}\cdot\mathbf{r}_i && \forall\ \langle \mathbf{V}_i, \mathbf{r}_i\rangle \in \mathit{GEN} \\ & && \gamma\mathbf{E}^\top\mathbf{f} + \boldsymbol{\beta} = \mathbf{0} \\ & && \mathbf{f} \geq \mathbf{0} \end{aligned}$$

3.4 Experiments

We assess the general performance of our approach using a set of randomly generated semi-sparse MDPs (as preceding section) and specific MDPs arising from the Autonomic Computing domain. All linear and mixed integer programming results described in this thesis were obtained using CPLEX 11 on PowerEdge 2950 servers with dual quad-core Intel E5355 CPUs.

To measure the performance of minimax regret computation, we first examine the constraint

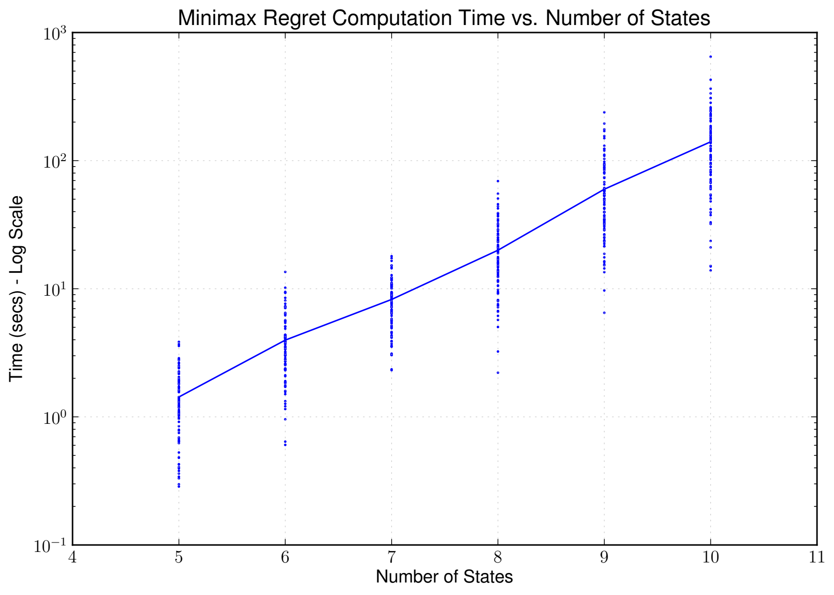


Figure 3.2: Scaling of constraint generation with number of states

generation procedure. Figure 3.1 plots the *regret gap* that measures the difference between the master problem value and subproblem value at each iteration versus the time (in milliseconds) to reach that iteration. Results are shown for 20 randomly generated MDPs with ten states and five actions.

Figure 3.2 shows how minimax regret computation time increases with the size of the MDP (5 actions, varying number of states). Constraint generation using the MIP formulation scales exponentially, hence computing minimax regret exactly is only feasible for small MDPs using this formulation. The next section will discuss an approximation that is far more efficient. Note that, the computations shown here are using the initial reward uncertainty. As queries refine the reward polytope, regret computation becomes faster in general.

In practice we have found that the constraint generation procedure requires relatively few iterations to converge; however, the computational cost per iteration can be high. This is due exclusively to the subproblem optimization, which requires the solution of a MIP with a large number of integer variables, one per state-action pair. The master problem optimization by contrast is extremely efficient. It is essentially solving the the standard MDP dual formulation

(2.12) with the addition of adversarial constraints. This suggests examination of approximations to the subproblem, i.e., the computation of max regret $MR(\mathbf{f}, \mathcal{R})$. These approximations will fit well in with elicitation as discussed further in Chapter 5.

3.5 Computing Approximate Max Regret

The high cost of computing exact minimax regret motivates the development of approximation methods. This section explore several approximation methods and assesses their performance on small and medium-sized MDPs. The goal of these methods is two-fold: 1) to quickly compute a solution to inform query selection heuristics; and 2) to generate a reasonable upper bound on minimax regret to serve as guarantee on regret for the user (in the event that elicitation is to be terminated). Methods for computing efficient lower bounds on minimax regret include a relaxation of the mixed integer programming (MIP) component of our constraint generation procedure and an alternating optimization scheme in which each component is independently optimized (holding other variables fixed). An efficient upper bound on minimax regret is found using the *reformulation-linearization technique* (RLT) of (Sherali and Alameddine, 1992).

3.5.1 Under-approximation

Linear Relaxation

We can dramatically improve the efficiency of the subproblem computation by relaxing all integrality constraints (3.18) on the binary policy indicators $\mathbf{I}_a$. The value function $\tilde{\mathbf{V}}$ resulting from this relaxation may not accurately reflect the (now potentially stochastic) adversarial policy. This is due to $\tilde{\mathbf{V}}$ including a fraction of the big-M term due to constraint (3.15): $\tilde{\mathbf{V}} \leq (1 - \mathbf{I}_a)\mathrm{M}_a + \mathbf{Q}_a$. However, the reward function $\mathbf{r}$ selected remains in the feasible set, and, empirically, the optimal value function $\mathbf{V}_\mathbf{r}$ w.r.t. reward $\mathbf{r}$ yields a solution to the subproblem that is close to optimal. Finding this optimal value function for $\mathbf{r}$ requires solving a standard MDP LP. Since the reward is a valid choice, this solution is guaranteed to be a lower bound on the solution to the subproblem.

Alternating Approximation

The alternating optimization procedure is a hill-climbing approach that iteratively computes an adversarial policy (for a fixed reward) and an adversarial reward (for a fixed policy). This reduces the max regret computation to the following sequence of LPs.
Given $\mathbf{f}, \mathbf{r}$:

$$\begin{aligned} \underset{\mathbf{g}}{\text{maximize}} \quad & \mathbf{g}\cdot\mathbf{r} - \mathbf{f}\cdot\mathbf{r} \\ \text{subject to:} \quad & \gamma\mathbf{E}^{\top}\mathbf{g} + \boldsymbol{\beta} = \mathbf{0},\ \mathbf{g} \geq \mathbf{0} \end{aligned} \tag{3.20}$$

Given $\mathbf{f}, \mathbf{g}$:

$$\begin{aligned} \underset{\mathbf{r}}{\text{maximize}} \quad & \mathbf{g}\cdot\mathbf{r} - \mathbf{f}\cdot\mathbf{r} \\ \text{subject to:} \quad & \mathbf{C}\mathbf{r} \leq \mathbf{d} \end{aligned} \tag{3.21}$$

The alternating optimization is essentially performing local search and does not guarantee a global optimum. However, at any iteration the adversary policy and reward are feasible and represent a valid lower bound on max regret.

3.5.2 Over-approximation

Reformulation-Linearization Technique (RLT)

The max regret optimization can be expressed directly in terms of the adversary policies (as occupancy frequencies) and the adversarial choice of reward function.

$$\begin{aligned} \underset{\mathbf{g},\mathbf{r}}{\text{maximize}} \quad & \mathbf{g}\cdot\mathbf{r} - \mathbf{f}\cdot\mathbf{r} \\ \text{subject to:} \quad & \gamma\mathbf{E}^{\top}\mathbf{g} + \boldsymbol{\beta} = \mathbf{0},\ \mathbf{g} \geq \mathbf{0} \\ & \mathbf{C}\mathbf{r} \leq \mathbf{d} \end{aligned} \tag{3.22}$$

Here the term $\mathbf{g}\cdot\mathbf{r}$ yields a non-convex bilinear program. We adopt a method from Sherali and Alameddine (1992) used for the global optimization of such non-convex bilinear programs.

The method reformulates the problem by constructing a set of variable factors using the problem constraints. Combinations of these factors are multiplied with the original problem constraints to generate additional valid nonlinear constraints. The resulting nonlinear program is then linearized by defining a new set of variables, one for each nonlinear term. The "RLT" process results in a linear program whose optimal value provides an upper bound on the optimal value to the original bilinear programming problem. Full details of the formulation can be found in Appendix C.1.

3.5.3 Approximating Minimax Regret

The described techniques for approximating max regret can be used to solve the subproblem component of constraint generation. However, convergence cannot be guaranteed in the absence of an exact subproblem solution. At any point during constraint generation, the solution to the master problem represents a valid lower bound on minimax regret. The upper bound on max regret generated by the RLT approach during constraint generation serves as an upper bound on minimax regret.

As we will discuss in the next section, the under-approximations exhibit a low error and can be used to efficiently guide reward elicitation (by providing an approximate minimax regret solution to inform query selection). The over-approximation provided by the RLT approach leads to significantly higher error, however, it offers a genuine upper bound on minimax regret

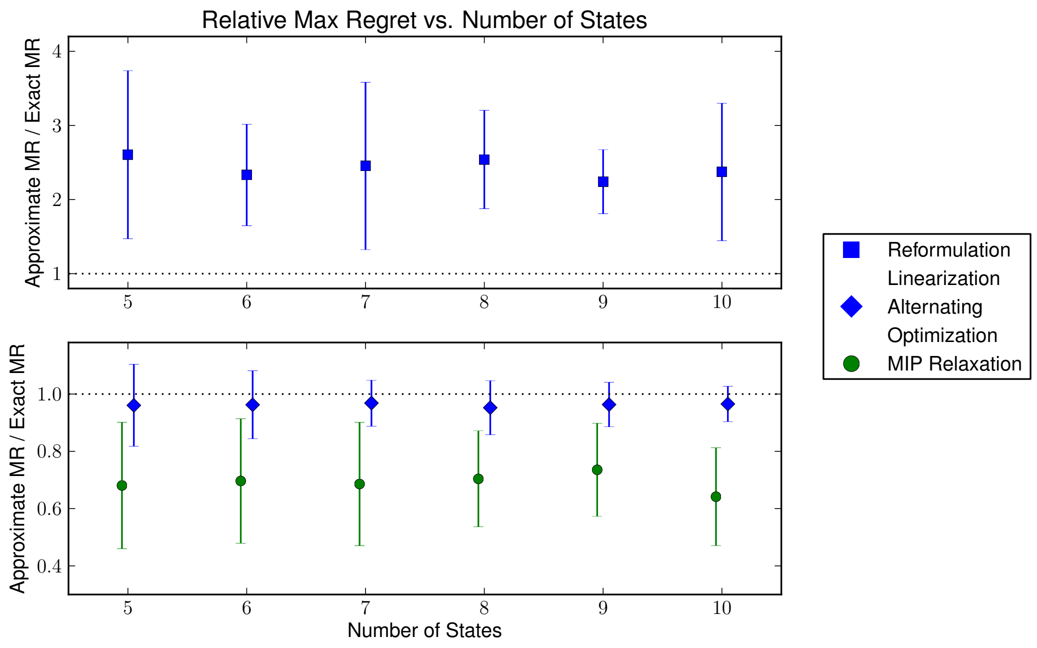


Figure 3.3: Relative (Approximate/True) Max Regret vs. Number of States. Results averaged over 50 runs.

that can be provided to the user. As elicitation proceeds we have observed that this upper bound is rendered tighter and tighter as reward uncertainty is reduced. Eventually the bound may drop low enough to meet a stopping criterion that stipulates minimax regret fall below a threshold. Experiments demonstrating this phenomena are detailed in Chapter 4 (after we introduce the prerequisite elicitation details).

3.5.4 Experiments

To evaluate the presented approximation schemes, we randomly generate semi-sparse IRMDPs (using the methodology from Section 3.2), while varying the number of states. Figure 3.3 shows average relative max regret (approximate MR / exact MR) along with the standard deviation over 50 runs. Figure 3.4 shows the average computation time in seconds over those runs.

The alternating approximation procedure performs extremely well, yielding an average rel-

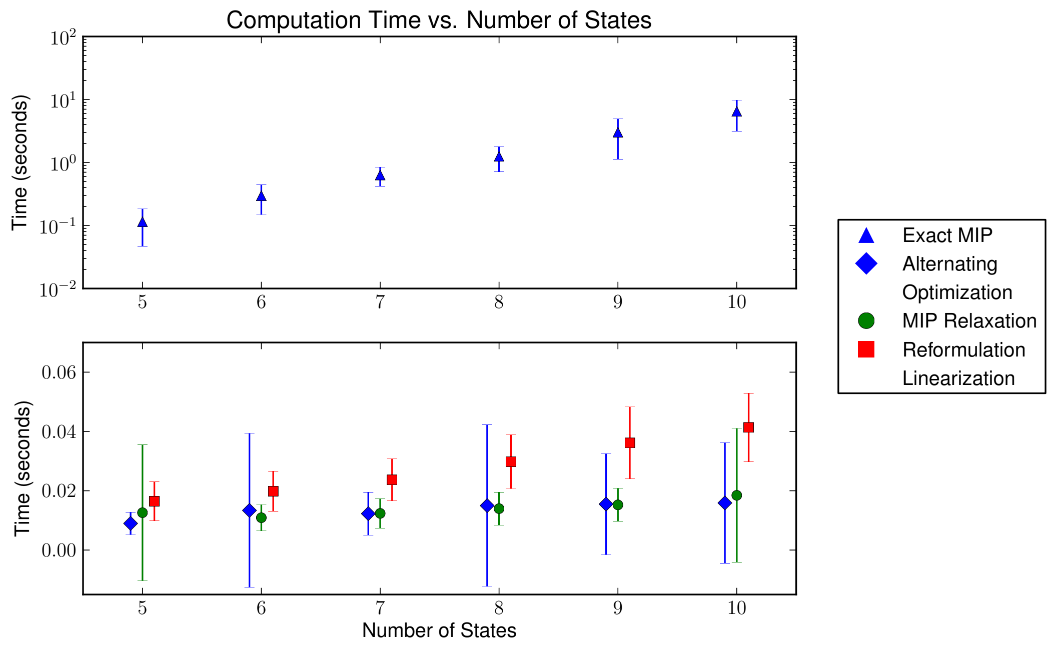


Figure 3.4: Time (in seconds) of Max Regret Computation vs. Number of States. Results averaged over 50 runs.

ative max regret between 0.952 and 0.968 (translating to a relative approximation error of 3.2–4.8%). The MIP relaxation is less effective, yielding an average relative max regret between 0.641 and 0.735 (translating to relative approximation error of 26.5–35.9%). Additionally, Figure 3.4 shows that the MIP relaxation is the least computationally efficient of the our approximation procedures.

The RLT procedure is the least effective, substantially over-approximating max regret with error as high as 390%; on average the measured relative max regret is between 2.33 and 2.60. Figure 3.4 shows that all of our approximation procedures offer substantial improvement in runtime over exact methods; for IRMDPs of ten states, approximate max regret on average is computed in less than 0.05 seconds using any of the three methods compared to 6.4 seconds for exact max regret computation using our mixed integer program.

We also evaluate the use of each approximation for computing minimax regret (replacing the MIP for computing the subproblem in our ICG procedure). Figure 3.5 shows the average

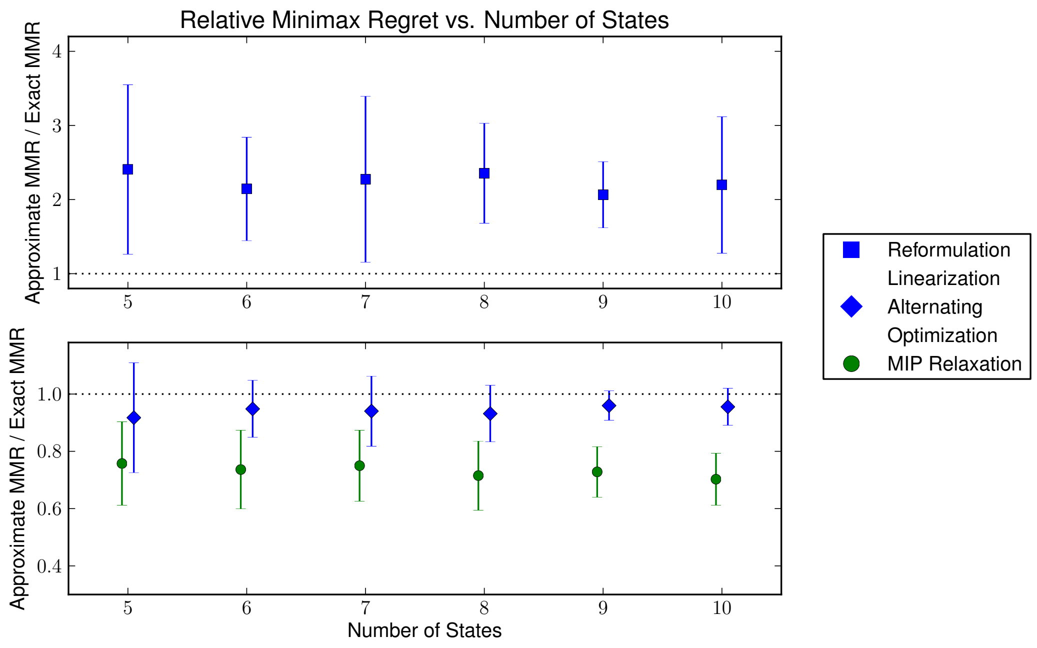


Figure 3.5: Relative (Approximate/True) Minimax Regret vs. Number of States. Results averaged over 50 runs.

relative minimax regret (approximate MMR / exact MMR) as the number of states in the randomly generated IRMDPs grow, while Figure 3.6 plots computational time. The approximation error observed in the minimax regret computation mirrors our results for the max regret computation. The alternating approximation continues to perform well offering relative minimax regret of between 0.917 and 0.959. In contrast, MIP relaxation resulted in relative minimax regret of between 0.702 and 0.757. An efficient upper bound on minimax regret is desirable for providing a guarantee to the user during reward elicitation; empirically our RLT procedure, while computationally efficient, offers a bound that is on average 2.06–2.40 times that of exact minimax regret. As elicitation progresses and reward uncertainty is reduced—in turn, reducing minimax regret—this upper bound is more able to meet a user's threshold for ending elicitation with a satisfactory level of regret (as we discuss in Chapter 6).

This suggests an approximate elicitation scheme where alternating approximation drives query selection and the upper bound derived from the RLT procedure provides a guarantee on regret (given the current uncertainty) and serves as a stopping criterion. After we introduce

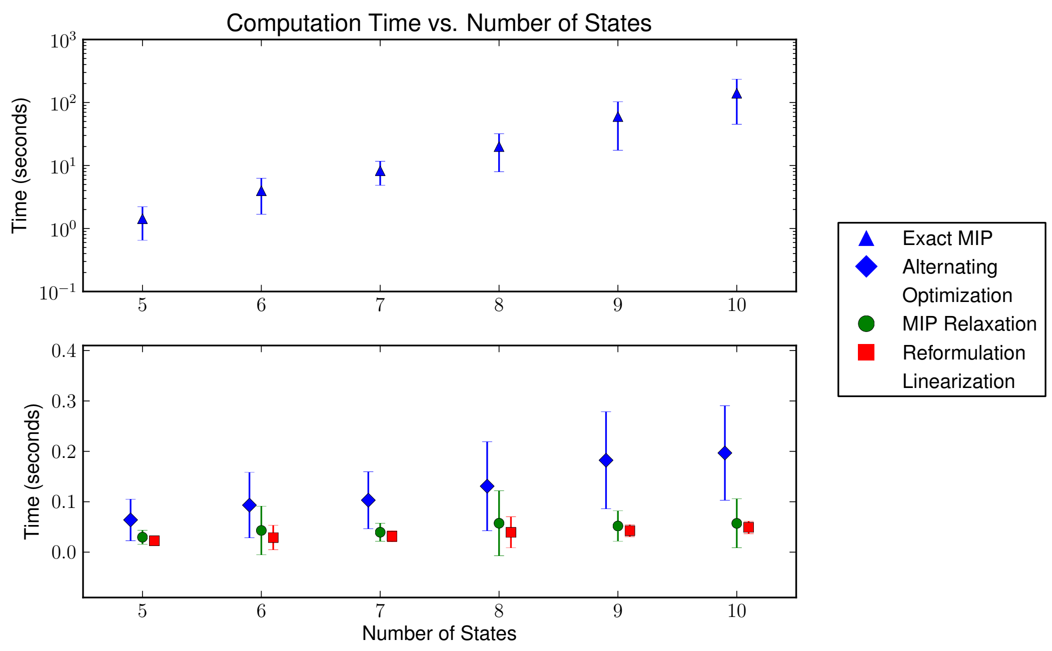


Figure 3.6: Time (in seconds) of Minimax Regret Computation vs. Number of States. Results averaged over 50 runs.

the core details of reward elicitation, we empirically examine this approximation approach for elicitation on random IRMDPs Section 4.2.3.

We need not always resort to approximation for fast minimax regret computation. Our discussion now shifts to an approach that exploits structure through a precomputation that facilitates significantly more efficient *online* computation during elicitation.

3.6 Leveraging Nondominated Policies

From the perspective of the adversary computing max regret, some policies are dominated and may safely be ignored. Formally we define a policy $\mathbf{f}$ to be *nondominated* w.r.t. to the feasible reward set $\mathcal{R}$ iff

$$\exists\, \mathbf{r} \in \mathcal{R} \text{ such that } \mathbf{f}\cdot\mathbf{r} \geq \mathbf{f}'\cdot\mathbf{r}, \qquad \forall\, \mathbf{f}' \in \mathcal{F} \tag{3.23}$$

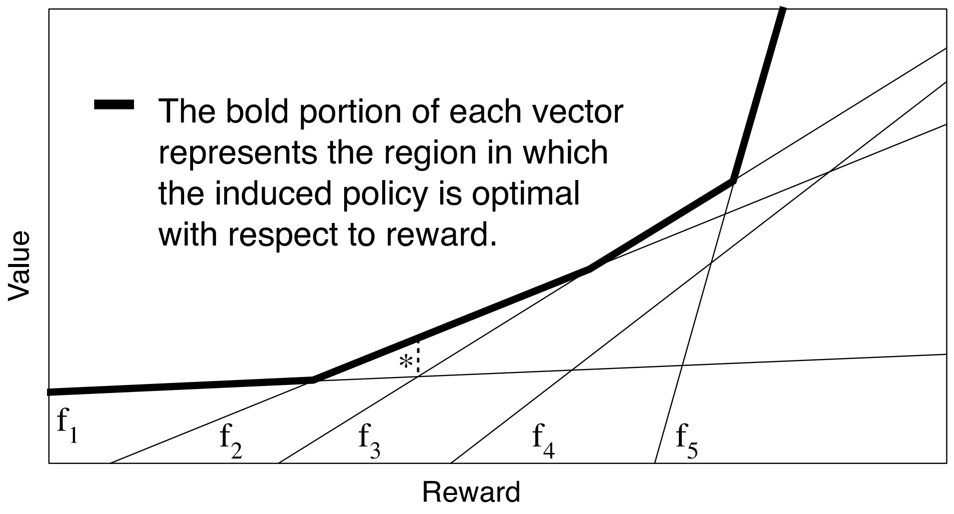


Figure 3.7: Illustration of the value of policies $f_1, \ldots, f_5$ as a linear function of one-dimensional uncertain reward.

In other words, a nondominated policy is optimal for some feasible reward. Let $\Gamma(\mathcal{R})$ denote the full set of nondominated policies w.r.t. $\mathcal{R}$. When $\mathcal{R}$ is fixed, we write Γ for simplicity and understand the "set of nondominated policies" to be implicitly defined w.r.t. the fixed $\mathcal{R}$.

Observation 1 *Given an IRMDP and policy* $\mathbf{f}$*:* $\operatorname{argmax}_{\mathbf{g}\in\mathcal{F}} PMR(\mathbf{f}, \mathbf{g}, \mathcal{R}) \in \Gamma$.

A brief proof of this observation can be found in Appendix A.1. It follows that the adversarial policy used to maximize regret of $\mathbf{f}$ must lie in Γ, since an adversary can only maximize regret by choosing some $\mathbf{r} \in \mathcal{R}$ and an optimal policy $\mathbf{f}^*_\mathbf{r}$ for $\mathbf{r}$.

If the set of nondominated policies is relatively small, and can be identified easily, then the complexity of minimax regret computation is greatly reduced. This section describes an algorithm for minimax regret computation that is polynomial in the number of nondominated policies and demonstrates its empirical advantage over related techniques (Xu and Mannor, 2009).

Define $\mathbb{V}(\mathbf{r}) = \max_{\mathbf{f}\in\mathcal{F}} \mathbf{f} \cdot \mathbf{r}$ to be the optimal value obtainable when $\mathbf{r} \in \mathcal{R}$ is the true reward. Since policy value is linear in $\mathbf{r}$, $\mathbb{V}$ is piecewise linear and convex (PWLC), much like the belief-state value function in POMDPs (Cheng, 1988; Kaelbling et al., 1998), a fact we exploit below. Figure 3.7 illustrates this for a simplified 1-D reward, with nondominated policy set $\Gamma = \{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3, \mathbf{f}_5\}$ ($\mathbf{f}_4$ is dominated, thus optimal for no reward).

Xu and Mannor (2009) propose a method that exploits nondominated policies, computing minimax regret using a linear program (LP), which "enumerates" Γ using $O(|\mathcal{R}||\Gamma|)$ variables. Given a set $\Gamma = \{\mathbf{f}_1, \dots, \mathbf{f}_t\}$ of t nondominated policies, the LP is expressed as follows:

$$\underset{\mathbf{z},\mathbf{c},\delta}{\text{minimize}} \quad \delta \tag{3.24}$$

$$\text{subject to:} \quad \sum_{i=1}^{t} c_i = 1$$

$$\mathbf{c} \geq \mathbf{0}$$

$$\left.\begin{array}{l} \delta \geq \mathbf{d}^\top \mathbf{z}(i) \\ \mathbf{C}^\top \mathbf{z}(i) + \hat{\Gamma}\mathbf{c} = \mathbf{f}_i \\ \mathbf{z}(i) \geq 0 \end{array}\right\} \quad i = 1, 2, \dots, t \tag{3.25}$$

where $\mathcal{R}$ is defined by inequalities $\mathbf{Cr} \leq \mathbf{d}$, and $\hat{\Gamma}$ is a matrix whose columns are elements of Γ. The t-dimensional vector $\mathbf{c}$ encodes a randomized policy with support set Γ, which the authors show must be minimax optimal. For each potential adversarial policy $\mathbf{f}_i \in \Gamma$, variable $\mathbf{z}(i)$ and Equations (3.25) encode the dual of the following formulation for computing pairwise max regret given a player policy $\mathbf{c}\hat{\Gamma}$ and adversary policy $\mathbf{f}_i$:

$$\underset{\mathbf{r}}{\text{maximize}} \quad \left(\mathbf{f}_i - \mathbf{c}\hat{\Gamma}\right) \cdot \mathbf{r}$$

$$\text{subject to:} \quad \mathbf{Cr} \leq \mathbf{d}$$

Thus minimizing $\mathbf{z}$ maximizes regret of the player policy encoded with $\mathbf{c}$. We refer to this approach as LP-ND1. Xu and Mannor provide no computational results for this formulation.

Rather than encoding the player's policy choice using a convex combination of all nondominated policies: $\mathcal{F} = \{\mathbf{c}\hat{\Gamma} \mid \mathbf{c} \geq 0, \sum_i c_i = 1\}$; we can modify the LP to encode the player's policy using the occupancy frequency constraints from Equation (2.11):

$\mathcal{F} = \{\mathbf{f} \mid \mathbf{f} \geq 0, \gamma\mathbf{E}^\top\mathbf{f} + \boldsymbol{\beta} = \mathbf{0}\}$. The following LP is identical to LP-ND1 save for the new player policy encoding:

$$\begin{aligned} \underset{\mathbf{z},\mathbf{f},\delta}{\text{minimize}} \quad & \delta \\ \text{subject to:} \quad & \gamma\mathbf{E}^\top\mathbf{f} + \boldsymbol{\beta} = \mathbf{0}, \mathbf{f} \geq \mathbf{0} \\ & \left.\begin{aligned} & \delta \geq \mathbf{d}^\top\mathbf{z}(i) \\ & \mathbf{C}^\top\mathbf{z}(i) + \mathbf{f} = \mathbf{f}_i \\ & \mathbf{z}(i) \geq 0 \end{aligned}\right\} i = 1, 2, \ldots, t \end{aligned} \tag{3.26}$$

This linear program, denoted LP-ND2, reduces the representation of the player's policy from $O(|\Gamma|)$ to $O(|S||A|)$ variables, which is advantageous when $|\Gamma| \gg |S||A|$ (as we will see below).

There is yet a third way to leverage nondominated policies to compute minimax regret. Rather than solving a single, large LP, we can use the constraint generation approach from Section 3.1 solving the master problem (3.19) as before:

$$\begin{aligned} \underset{\mathbf{f},\delta}{\text{minimize}} \quad & \delta \\ \text{subject to:} \quad & \delta \geq \mathbf{g}_i \cdot \mathbf{r}_i - \mathbf{f} \cdot \mathbf{r}_i & \forall \langle \mathbf{g}_i, \mathbf{r}_i \rangle \in \mathit{GEN} \\ & \gamma\mathbf{E}^\top\mathbf{f} + \boldsymbol{\beta} = \mathbf{0} \\ & \mathbf{f} \geq \mathbf{0} \end{aligned}$$

However, rather than resorting to the MIP (3.14) we instead exploit Obs. 1 and solve, for each $\mathbf{g} \in \Gamma$, a small LP to determine the reward that gives $\mathbf{g}$ maximal advantage over the current relaxed solution $\mathbf{f}$:

$$\begin{aligned} \underset{\mathbf{r}}{\text{maximize}} \quad & \mathbf{g} \cdot \mathbf{r} - \mathbf{f} \cdot \mathbf{r} \\ \text{subject to:} \quad & A\mathbf{r} \leq \mathbf{d} \end{aligned}$$

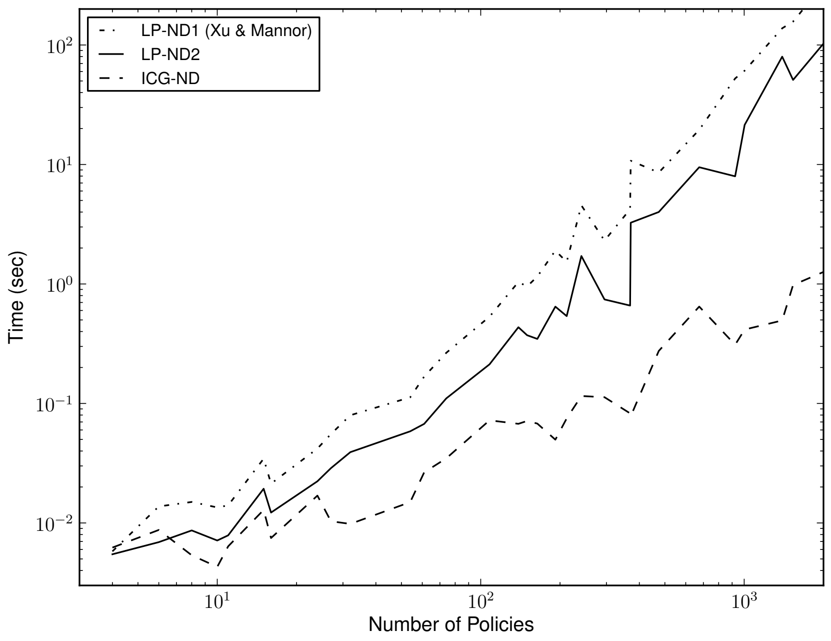


Figure 3.8: Scaling of MMR computation w.r.t. nondominated policies

The g with largest objective value determines the maximally violated constraint. Thus the MIP computing max regret may be replaced with a set of smaller LPs. We denote this approach ICG-ND.

3.6.1 Experiments

We compare these three approaches to minimax regret computation using nondominated policies, as well as the MIP-approach described in Section 3.1, here denoted ICG-MIP. We randomly generate small semi-sparse IRMDPs using the procedure from Section 3.2, while fixing $|A| = 5$ and varying the number of states from 3 to 7. We generate 20 MDPs of each size.

Figure 3.8 shows the computation time of the different algorithms as a function of the number of nondominated policies in each sampled IRMDP. LP-ND1 (Xu and Mannor, 2009) performs poorly, taking more than 100 seconds to compute minimax regret for IRMDPs with more than 1000 nondominated policies. Our modified LP, LP-ND2, performs only slightly

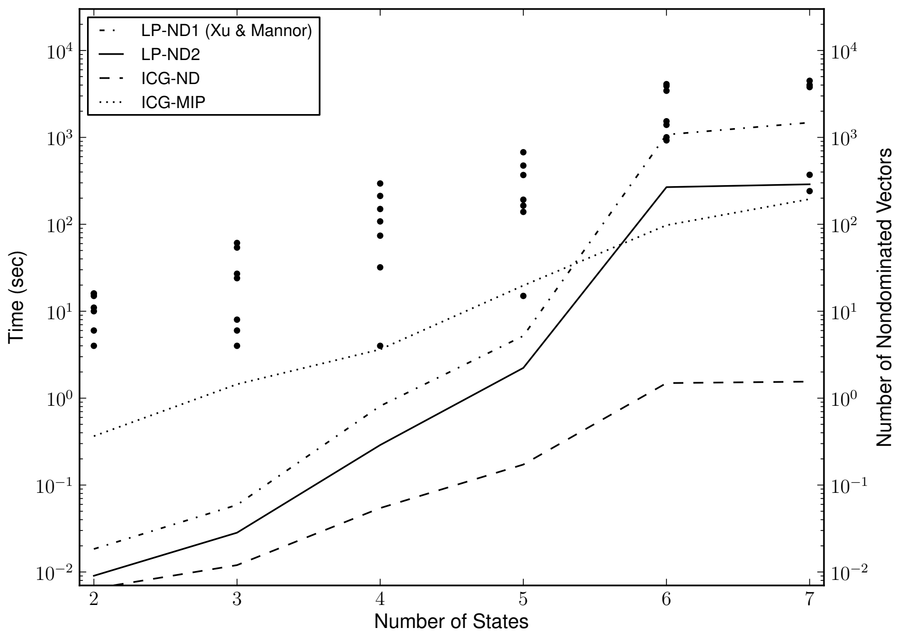


Figure 3.9: Scaling of MMR computation (lineplot on left y-axis) and nondominated policies (scatterplot on right y-axis) w.r.t. number of states

better. The most effective approach is our LP-based constraint generation procedure, ICG-ND, in which nondominated policies are exploited to determine maximally violated constraints. While $|\Gamma|$ LPs must be solved at each iteration, these are relatively small, containing far fewer variables and constraints than LP-ND1 and LP-ND2. ICG-ND is also more effective than the original MIP model ICG-MIP, which does not make use of nondominated policies. This is seen in Figure 3.9, which shows average computation time (lines) and number of nondominated vectors (scatterplot) for each MDP size. While ICG-MIP performs reasonably well as the number of states grows (eventually outperforming LP-ND1 and LP-ND2), the ICG-ND approach still takes roughly an order of magnitude less time than ICG-MIP. As a result, the proceeding approaches to working with larger MDPs will build upon the iterative constraint generation approach to leveraging nondominated policies. Next, we shift focus to identifying nondominated policies.

3.7 Generating Nondominated Policies using the πWitness Algorithm

The effectiveness of the iterative constraint generation procedure in exploiting the nondominated set Γ seems evident, but it relies on identifying the set of nondominated policies. There is a connection between our use of policies that are nondominated w.r.t. uncertain reward and algorithms for partially observable MDPs (POMDPs) that compute and leverage the set of policies that are nondominated w.r.t. to uncertainty over states. This section examines this connection and following the work of Kaelbling et al. (1998) develops the π*Witness algorithm* for generating Γ, the set of nondominated policies. The πWitness algorithm allows for Γ to be computed offline. When the number of policies generated is small, this results in efficient online computation of minimax regret. Chapter 6 will return to the discussion of nondominated policies, and will focus on computing small approximate sets of nondominated policies (to ensure efficient computation) while maintaining a bound on the error resulting in the effective approximation of minimax regret.

3.7.1 The πWitness Algorithm

Crucial to the operation of the πWitness algorithm is the concept of *local adjustments* of the occupancy frequencies $\mathbf{f}$ corresponding to a policy π. Suppose, when starting at state s we take action a rather than $\pi(s)$ as prescribed by π, but follow π thereafter. The occupancy frequencies induced by this *local adjustment* to π are given by:

$$\mathbf{f}^{s:a} = \beta(s)(\mathbf{e}_{sa} + \gamma \sum_{s'} \Pr(s'|s,a)\mathbf{f}[s']) + (1 - \beta(s))\mathbf{f}$$

where $\mathbf{e}_{sa}$ is an $S \times A$ vector with a 1 in position s, a and zeroes elsewhere. $\mathbf{f}[s]$ denotes the occupancy frequencies induced by starting in state s (ignoring the starting state distribution β). It follows from standard policy improvement theorems (Puterman, 1994) that if $\mathbf{f}$ is not optimal

Algorithm 3: The πWitness algorithm

```
$\mathbf{r} \leftarrow$ some arbitrary $\mathbf{r} \in \mathcal{R}$
$\mathbf{f} \leftarrow$ findBest($\mathbf{r}$)
$\Gamma \leftarrow \{ \mathbf{f} \}$
agenda $\leftarrow \{ \mathbf{f} \}$
while agenda is not empty do
    $\mathbf{f} \leftarrow$ next item in agenda
    foreach $s, a$ do
        $\delta$, $\mathbf{r}^w \leftarrow$ findWitnessReward($\mathbf{f}^{s:a}, \Gamma$)
        while $\delta > 0$ (witness found) do
            $\mathbf{f}_{best} \leftarrow$ findBest($\mathbf{r}^w$)
            add $\mathbf{f}_{best}$ to $\Gamma$
            add $\mathbf{f}_{best}$ to agenda
            $\delta$,$\mathbf{r}^w \leftarrow$ findWitnessReward($\mathbf{f}^{s:a}, \Gamma$)
        end
    end
end
```

for reward $\mathbf{r}$, then there must be a local adjustment (s, a) such that $\mathbf{f}^{s:a} \cdot \mathbf{r} > \mathbf{f} \cdot \mathbf{r}$.[1]

Theorem 1 *Let $\Gamma' \subsetneq \Gamma$ be a (strictly) partial set of nondominated policies. Then there is an $\mathbf{f} \in \Gamma'$, an (s, a), and an $\mathbf{r} \in \mathcal{R}$ such that $\mathbf{f}^{s:a} \cdot \mathbf{r} > \mathbf{f}' \cdot \mathbf{r}, \quad \forall\, \mathbf{f}' \in \Gamma'$.*

This theorem is analogous to the witness theorem for POMDPs (Kaelbling et al., 1998). Details of the proof can be found in Appendix A.2. This theorem underpins a Witness-style algorithm for computing Γ. Our πWitness algorithm begins with a partial set Γ consisting of a single nondominated policy, namely a policy that is optimal for an arbitrary $\mathbf{r} \in \mathcal{R}$. At each iteration, for all $\mathbf{f} \in \Gamma$, it checks whether there is a local adjustment (s, a) and a witness reward $\mathbf{r}$ such that $\mathbf{f}^{s:a} \cdot \mathbf{r} > \mathbf{f}' \cdot \mathbf{r}$ for all $\mathbf{f}' \in \Gamma$ (determining whether $\mathbf{f}^{s:a}$ offers an improvement at $\mathbf{r}$). If there is an improvement, we add the optimal policy $\mathbf{f}^*_{\mathbf{r}}$ for that $\mathbf{r}$ to Γ. If no improvement exists for any $\mathbf{f}$, then by Theorem 1, Γ is complete. The procedure is detailed in Algorithm 3. We define the *agenda* to hold the policies for which we have not yet explored all local adjustments. The subroutine **findWitnessReward** searches for a reward $\mathbf{r}$ at which $\mathbf{f}^{s:a}$ has higher value than

[1] For ease of exposition β is assumed to be is strictly positive. Our definitions are easily modified if $\beta(s) = 0$ for some s.

any $\mathbf{f}' \in \Gamma$ by solving the following LP:

$$\left[\textbf{findWitnessReward}\right] \qquad \begin{array}{lll} \underset{\delta,\mathbf{r}}{\text{maximize}} & \delta & \\ \text{subject to:} & \delta \leq \mathbf{f}^{s:a}\cdot\mathbf{r} - \mathbf{f}'\cdot\mathbf{r} & \forall\, \mathbf{f}' \in \Gamma \\ & \mathbf{C}\mathbf{r} \leq \mathbf{d} & \end{array}$$

findWitnessReward returns the optimal values for $\mathbf{r}$ and δ. The improvement offered by policy $\mathbf{f}^{s:a}$ over the best policy in Γ (given $\mathbf{r}$) is quantified by δ; when $\delta \leq 0$, there is no witness reward at which $\mathbf{f}^{s:a}$ offers improvement. There may be multiple witnesses for a single adjustment, thus **findWitnessReward** is called until no more witnesses are found. The subroutine **findBest** finds the optimal policy given $\mathbf{r}$ and can be implemented using any of the standard MDP solution algorithm described in Section 2.2.2. We use the following LP:

$$\left[\textbf{findBest}\right] \qquad \begin{array}{ll} \underset{\mathbf{f}}{\text{maximize}} & \mathbf{f}\cdot\mathbf{r} \\ \text{subject to:} & \gamma\mathbf{E}^{\top}\mathbf{f} + \boldsymbol{\beta} = \mathbf{0}, \mathbf{f} \geq \mathbf{0} \end{array}$$

findBest returns the optimal policy $\mathbf{f}$. The runtime of the πWitness algorithm is polynomial in inputs $|S|$, $|A|$, $|\mathcal{R}|$, and output $|\Gamma|$, assuming bounded precision in the input representation. When a witness $\mathbf{r}^w$ is found by the algorithm, it testifies to a nondominated $\mathbf{f}$ which is added to Γ and to the agenda. Thus, the number of policies added to the agenda is exactly $|\Gamma|$. The subroutine **findWitnessReward** is called at most $|S||A|$ times for each $\mathbf{f} \in |\Gamma|$ to test local adjustments for witness points (for a total of $|\Gamma||S||A|$ calls). **findWitnessReward** requires solution of an LP with $|S||A| + 1$ variables and no more than $|\Gamma| + |\mathcal{R}|$ constraints, thus the LP encoding has polynomial size (hence solvable in polytime). The subroutine **findBest** is called only when a witness is found, and thus is used exactly $|\Gamma|$ times. It requires solving an MDP, which is polynomial in the size of its specification (Puterman, 1994). Thus πWitness is polynomial. This also means that for any class of MDPs with a polynomial number of

State Size	Number of Vectors		πWitness Runtime (secs)	
	μ	σ	μ	σ
4	3.463	2.231	0.064	0.045
8	3.772	3.189	0.145	0.144
16	7.157	5.743	0.433	0.329
32	7.953	6.997	1.228	1.062
64	11.251	9.349	4.883	3.981

Table 3.1: Varying Number of States

nondominated policies, minimax regret computation is itself polynomial.

3.7.2 Empirical Results

The number of nondominated policies $|\Gamma|$ is influenced largely by the dimensionality of the reward function and less so by conventional measures of MDP size, $|S|$ and $|A|$. A high dimensional $\mathbf{r}$ allows variability across the state-action space, admitting different optimal policies depending on the realization of reward. When reward is completely unrestricted (i.e., reward points $r(s, a)$ are "independent") our results from Section 3.6.1 (Figure 3.9) demonstrate that even small MDPs can admit a huge number of nondominated policies. However, in practice, reward functions typically have significant structure. Factored MDPs (Boutilier et al., 1999) have large state and action spaces defined over sets of state variables. Typically reward depends only on a small fraction of these, often in an additive way. In our empirical investigation of πWitness, we exploit this fact, exploring how its performance varies with reward dimension. Chapter 5 will revisit factored IRMDPs to describe how additive structure in reward functions can be leveraged to streamline elicitation.

To examine the link between factored MDP structure and $|\Gamma|$, we generated the full set of nondominated policies for IRMDPs of varying sizes, but with reward of small fixed dimension. States are defined by 2–6 binary variables (yielding $|S| = 4, \ldots, 64$), and a factored additive reward function on two attributes: $r(\mathbf{s}) = r_1(x_1) + r(x_2)$. The transition model and feasible reward set $\mathcal{R}$ is generated randomly as in Section 3.2, with random reward intervals generated for the parameters of each factor rather than for each (s, a)-pair. We continue to use an

Reward Dim.	Number of Vectors		πWitness Runtime (secs)	
	μ	σ	μ	σ
2	2.050	0.887	1.093	0.634
4	10.20	10.05	4.554	4.483
6	759.6	707.4	1178	1660
8	6116	5514	80642	77635

Table 3.2: Varying Dimension of Reward Space

unstructured transition model to emphasize the dependence on reward dimensionality.

Table 3.1 shows the number of nondominated policies discovered (with mean (μ) and standard deviation (σ) over 20 runs), and demonstrates that Γ does not grow appreciably with $|S|$, as expected with 2-D reward. The running time of nondominated policy generation (using πWitness) is similar, growing slightly greater than linearly in $|S|$.

We also examine MDPs of fixed size (6 attributes, $|S| = 64$), varying the dimensionality of the reward function from 2–8 by varying the number of additive reward attributes from 1–4. Results (20 instances of each dimension) are shown Table 3.2. While Γ is very small for dimensions 2 and 4, it grows dramatically with reward dimensionality, as does the running time of policy generation. This demonstrates the strong impact of the size of the output set Γ on the running time of nondominated policy generation procedure.

3.7.3 Approximating the Nondominated Set

The complexity of both πWitness and our ICG-ND procedure for computing minimax regret are influenced heavily by the size of Γ. While the number of nondominated policies remains manageable with as we scale the MDP state and action space, Γ grows quickly as we increase reward dimensionality. This motivates investigation of methods that use only a subset of the nondominated policies that reasonably approximate Γ, or specifically, the PWLC value function induced by Γ. We first explore theoretical guarantees on minimax regret when ICG-ND (or any other method that exploits Γ) is run using a subset of Γ.

Let $\tilde{\Gamma} \subseteq \Gamma$ and let the value function induced by $\tilde{\Gamma}$ w.r.t. to the fixed reward $\mathbf{r}$ be defined as:

$$\mathbb{V}_{\tilde{\Gamma}}(\mathbf{r}) = \max_{\mathbf{f} \in \tilde{\Gamma}} \ \mathbf{f} \cdot \mathbf{r} \tag{3.27}$$

Note that $\mathbb{V}_{\tilde{\Gamma}}(\mathbf{r})$ forms a lower bound on $\mathbb{V}_{\Gamma}(\mathbf{r})$. Define the *error* in $\mathbb{V}_{\tilde{\Gamma}}$ to be maximum difference between the approximate and exact value functions w.r.t. reward:

$$\epsilon_{\mathbb{V}}(\tilde{\Gamma}) = \max_{\mathbf{r} \in \mathcal{R}} \mathbb{V}_{\Gamma}(\mathbf{r}) - \mathbb{V}_{\tilde{\Gamma}}(\mathbf{r})$$

This error is illustrated in Figure 3.7, where the dashed line (marked with a *) shows the error introduced by using the subset of dominated policies $\{\mathbf{f}_1, \mathbf{f}_3, \mathbf{f}_5\}$ (removing $\mathbf{f}_2$). The error in $\mathbb{V}_{\tilde{\Gamma}}$ can be used to derive a bound on error in computed minimax regret. Let $MMR(\Gamma)$ denote true minimax regret when adversarial policy choice is unrestricted and $MMR(\tilde{\Gamma})$ denote the approximation when adversarial choice is restricted to $\tilde{\Gamma}$.[1] $MMR(\tilde{\Gamma})$ offers a lower bound on true MMR and the difference, denoted $\epsilon_{MMR}(\tilde{\Gamma})$, can be bounded:

$$\epsilon_{MMR}(\tilde{\Gamma}) = MMR(\Gamma) - MMR(\tilde{\Gamma}) \leq \epsilon_{\mathbb{V}}(\tilde{\Gamma}) \tag{3.28}$$

Let $\mathbf{f}_{\Gamma}$ and $\mathbf{f}_{\tilde{\Gamma}}$ be the minimax regret optimal 'player" policies when the adversary is restricted to policies from Γ and $\tilde{\Gamma}$ respectively. Let $MR(\mathbf{f}, \Gamma')$ be the max regret of policy $\mathbf{f}$ when the adversary's choice of policies is restricted to $\Gamma' \subseteq \Gamma$. Then by definition $MR(\mathbf{f}_{\Gamma}, \Gamma) = MMR(\Gamma)$ and $MR(\mathbf{f}_{\tilde{\Gamma}}, \tilde{\Gamma}) = MMR(\tilde{\Gamma})$. If we fix $\mathbf{f}_{\tilde{\Gamma}}$ and relax the constraints on the adversary's choice to allow any policy in Γ, then the maximum amount that the adversary can increase regret is bounded by $\epsilon_{\mathbb{V}}(\tilde{\Gamma})$:

$$MR(\mathbf{f}_{\tilde{\Gamma}}, \Gamma) - MR(\mathbf{f}_{\tilde{\Gamma}}, \tilde{\Gamma}) \leq \epsilon_{\mathbb{V}}(\tilde{\Gamma}),$$

[1]This does not depend on the algorithm used to compute MMR.

If the player is allowed to vary their policy to minimize regret against the unconstrained adversary—choosing $\mathbf{f} = \operatorname{argmin}_{\mathbf{f}} MR(\mathbf{f}, \Gamma)$—it will only decrease regret. Thus $MR(\mathbf{f}_\Gamma, \Gamma) - MR(\mathbf{f}_{\tilde{\Gamma}}, \tilde{\Gamma}) \leq \epsilon_{\mathbb{V}}(\tilde{\Gamma})$ which is equivalent to expression (3.28): $MMR(\Gamma) - MMR(\tilde{\Gamma}) \leq \epsilon_{\mathbb{V}}(\tilde{\Gamma})$.

The difference between the exact max regret of the approximately optimal policy $\mathbf{f}_{\tilde{\Gamma}}$ and exact minimax regret can also be bounded:

$$MR(\mathbf{f}_{\tilde{\Gamma}}, \Gamma) - MMR(\Gamma) \leq \epsilon_{\mathbb{V}}(\tilde{\Gamma}) \tag{3.29}$$

This bound can be established from the following observation. The exact max regret of $\mathbf{f}_{\tilde{\Gamma}}$ must be at least as great as the exact max regret of $\mathbf{f}_\Gamma$, since by definition $\mathbf{f}_\Gamma$ minimizes max regret. Thus $MR(\mathbf{f}_{\tilde{\Gamma}}, \Gamma) \geq MR(\mathbf{f}_\Gamma, \Gamma)$. We again examine $MR(\mathbf{f}_{\tilde{\Gamma}}, \Gamma)$, given a fixed player policy $\mathbf{f}_{\tilde{\Gamma}}$, and observe that restricting the adversary to policies from $\tilde{\Gamma}$ when computing max regret will reduce regret by at most $\epsilon_{\mathbb{V}}(\tilde{\Gamma})$—compared to an unrestricted adversary—yielding: $MR(\mathbf{f}_{\tilde{\Gamma}}, \Gamma) - MR(\mathbf{f}_{\tilde{\Gamma}}, \tilde{\Gamma}) \leq \epsilon_{\mathbb{V}}(\tilde{\Gamma})$. Finally, bound (3.28) implies: $MR(\mathbf{f}_\Gamma, \Gamma) - MR(\mathbf{f}_{\tilde{\Gamma}}, \tilde{\Gamma}) \leq \epsilon_{\mathbb{V}}(\tilde{\Gamma})$. Taken together these observations imply our bound:

$$\begin{aligned}
&MR(\mathbf{f}_{\tilde{\Gamma}}, \Gamma) \geq MR(\mathbf{f}_\Gamma, \Gamma) \\
&MR(\mathbf{f}_{\tilde{\Gamma}}, \Gamma) - MR(\mathbf{f}_{\tilde{\Gamma}}, \tilde{\Gamma}) \leq \epsilon_{\mathbb{V}}(\tilde{\Gamma}) \\
&MR(\mathbf{f}_\Gamma, \Gamma) - MR(\mathbf{f}_{\tilde{\Gamma}}, \tilde{\Gamma}) \leq \epsilon_{\mathbb{V}}(\tilde{\Gamma}) \\
\therefore \quad &MR(\mathbf{f}_{\tilde{\Gamma}}, \Gamma) - MR(\mathbf{f}_\Gamma, \Gamma) \leq \epsilon_{\mathbb{V}}(\tilde{\Gamma}) \Leftrightarrow \\
&MR(\mathbf{f}_{\tilde{\Gamma}}, \Gamma) - MMR(\Gamma) \leq \epsilon_{\mathbb{V}}(\tilde{\Gamma})
\end{aligned}$$

Should we generate a set of nondominated policies $\tilde{\Gamma}$ that ϵ-approximates Γ, any algorithm (including ICG-ND) that uses nondominated sets will produce a policy that is within a factor of $\epsilon_{\mathbb{V}}(\tilde{\Gamma})$ of minimizing max regret.

By carefully adding policies to Γ that contribute the most to error reduction, we may be able to construct a partial set $\tilde{\Gamma}$ of small size that closely approximates Γ. Our immediate discussion will focus on how the agenda in πWitness can be managed to better accommodate perspicacious error reduction. Section 6.2 will further the discussion by introducing the NRV algorithm which seeks to generate nondominated polices in an anytime fashion so as to maximally reduce the error at each step.

πWitness Anytime Performance We can construct a small approximating set $\tilde{\Gamma}$ using πWitness by exploiting its anytime properties and careful management of the agenda. Intuitively, we want to add policies to $\tilde{\Gamma}$ that hold the greatest promise for reducing error $\epsilon_{\mathbb{V}}(\tilde{\Gamma})$. We quantify this potential reduction as follows. Let $\tilde{\Gamma}_n$ be the nth nondominated set produced by πWitness, constructed by adding optimal policy $\mathbf{f}^*_n$ for the nth witness point $\mathbf{r}_n$. When $\mathbf{f}^*_n$ is added to the agenda, it offers improvement to the current approximation:

$$\Delta\mathbb{V}(\mathbf{f}^*_n) = \mathbb{V}_{\tilde{\Gamma}_n}(\mathbf{r}_n) - \mathbb{V}_{\tilde{\Gamma}_{n-1}}(\mathbf{r}_n).$$

We process the agenda in priority queue fashion, using $\Delta\mathbb{V}(\mathbf{f})$ as the priority measure for any policy $\mathbf{f}$ remaining on the agenda. Thus, we examine adjustments to policies that provided greater increase in value when added to $\tilde{\Gamma}$ before considering adjustments to policies that provided lesser value. Chapter 6 will discuss a more principled approach to generating nondominated policies in an online settings using a variant of Cheng's Linear Support algorithm (Cheng, 1988).

Informal experiments show that using a priority queue reduced the error $\epsilon_{\mathbb{V}}(\tilde{\Gamma})$ much more quickly than using standard stack or queue approaches. Hence we investigate the anytime performance of πWitness with a priority queue on random MDPs with 128 and 256 states (30 runs of each). The reward dimension is fixed to 6 (3 additive binary factors) and the number of actions to 5. We first compute the exact minimax regret for the MDP, then run πWitness. When the nth nondominated policy is found, we compute an approximation of minimax regret using the algorithm ICG-ND with approximate nondominated set Γ_n. We measure the relative error in minimax regret: $\epsilon_{MMR}(\tilde{\Gamma})/MMR$.

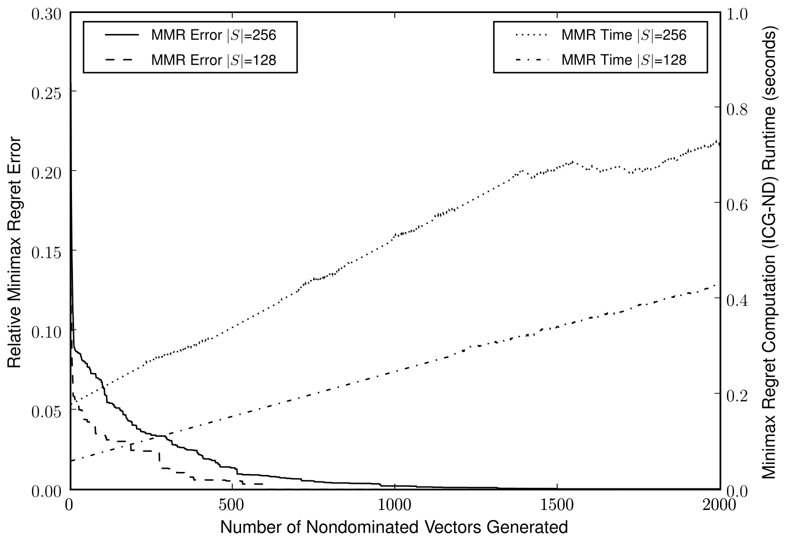


Figure 3.10: Relative minimax regret error and cumulative πWitness runtime vs. number of nondominated policies.

Figure 3.10 shows the relative error as nondominated policies are added using the priority queue implementation. The runtime of ICG-ND algorithm for computing minimax regret is also shown. On IRMDPs with 256 states, relative error drops below 0.02 after 500 policies have been added to Γ. Respectively, with 128 states, relative error drops below 0.02 after just 300 policies.

Minimax regret computation using ICG-ND grows linearly with the number of nondominated policies added to $|\Gamma|$, but stays well below 1 second: at the 0.02 error point, solution of 256-state (resp., 128-state) MDPs averages under 0.4 seconds (resp., 0.2 seconds). Given our goal of using minimax regret to drive preference elicitation, these results indicate that using a small set of nondominated policies and the ICG-ND algorithm will allow for real-time interaction with users.

While πWitness is much more computationally intensive, it can be run offline, once, to precompute nondominated policies (or a small approximate set) before engaging in online elicitation with users. Figure 3.11 shows the cumulative runtime of πWitness as it adds policies to Γ. With 256 states, the first 500 policies (yielding an empirical error level of 0.02) are

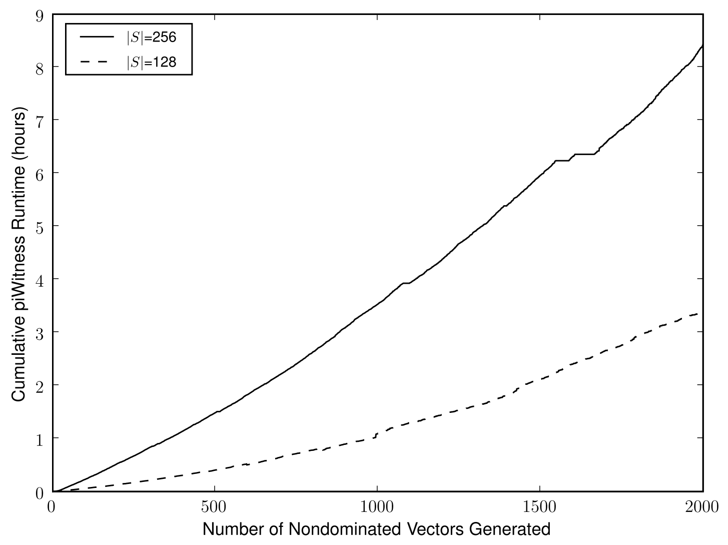


Figure 3.11: πWitness computation time (hrs.) vs. number of nondominated policies.

generated in under 2 hours on average. For 128 states, this is accomplished in under 1 hour. In both cases, runtime πWitness is only slightly super-linear in the number of policies.

3.8 Summary and Conclusions

Previous work has addressed reward uncertainty in Markov decision processes (detailed fully in Section 2.3). Each approach can be distinguished by the criterion used to compute policies and the assumptions made about the nature of reward uncertainty. For instance, the work of Delage and Mannor (2007) adopts the percentile criterion and assume that the prior probability of each realizable reward function is available. McMahan, Gordon and Blum (2003) compute policies that optimize maximin value for reward functions with strict uncertainty. Methods from inverse reinforcement learning establish constraints on the reward function by observing user behaviour. User behaviour can be interpreted as placing hard constraints on potential reward functions, giving rise to criteria that maximize the margin to these constraints (Ratliff et al., 2006) or the margin to other sub-optimal functions (Ng and Russell, 2000). Alternately, probabilistic interpretations of user behaviour have been used to compute policies that minimize expected loss (Ramachandran and Amir, 2007) or maximize entropy (Ziebart et al., 2008).

The work presented in chapter is the first to suggest applying the minimax regret criterion to MDPs with strict reward uncertainty. Minimax regret is an especially suitable measure in the context of elicitation, since it can guide query selection and offer intuitive guarantees to the user. Section 3.1 offers—to our knowledge—the first approach to minimax regret computation for IRMDPs. However, the mixed integer programming at the heart of this approach forms a computational bottleneck, limiting its usefulness to small IRMDPs. Given the computational intractability of computing minimax regret for general IRMDPs (Xu and Mannor, 2009) this limitation is unsurprising.

We develop several computationally efficient approximations. Empirically, the alternating optimization approximation offers low error and can be used to inform query selection, however, as an under-approximation it does not yield an upper bound on minimax regret that could provide a guarantee to the user. The reformulation-linearization (RLT) approximation offers the desired upper bound, but empirically exhibits relatively high error. We are unaware of any related approaches to approximate minimax regret for IRMDPs. Further investigation of methods beyond RLT from the global optimization literature (Torn and Zilinskas, 1989) may yield improved upper bounds.

The use of nondominated policies for minimax regret computation was simultaneously suggested by Xu and Mannor (2009), and Regan and Boutilier (2009; 2010). Section 3.6 demonstrates that leveraging the set Γ of nondominated policies can yield orders of magnitude improvement in minimax regret computation time. To generate the set c, we introduce the πWitness algorithm. The complexity of both nondominated policy generation and minimax regret computation is tightly tied to the cardinality of Γ which we observe to be related to the dimension of the reward function. Thus, IRMDPs with large state spaces and compact reward functions can be tackled efficiently, however, as reward dimensionality grows, an exact representation of Γ becomes less useful. Section 3.7.3 begins our investigation of anytime approaches for generating approximate sets Γ; Chapter 6 revisits this topic, offering an improved algorithm with anytime error bounds. The related work of Oh & Kim (Oh and Kim, 2011b) develops both exact and anytime algorithms for generating Γ by partitioning reward space using the inverse reinforcement learning constraints. We empirically examine the efficacy of these algorithms in conjunction with our own work in Section 6.3.

We briefly offer guidance to selecting among the variety of approaches to minimax regret computation that have been presented in this chapter. For small IRMDPs the method of constraint generation and mixed integer programming presented in Section 3.1 is a good choice; offers an exact solution and does not require offline pre-computation.

For larger IRMDPs, we recommend exploring potential problem structure by first generating the set Γ of nondominated policies offline. If the generated set of nondominated policies is small it can be used to quickly compute exact minimax regret online during reward elicitation. When the cardinality of Γ if too large to support efficient exact minimax regret computation, we can resort to approximating minimax regret using partial sets of nondominated policies. The πWitness algorithm detailed in Section 3.7.1 can be used in an anytime fashion, and empirically the approximation error drops quickly as policies are generated. However, the NRV algorithm that will be presented in Section 6.2 holds more promise for generating partial sets of nondominated policies since it provides an anytime bound on approximation error and operates by generating new nondominated policies to directly reduce this error. For large IRMDPs the offline generation of nondominated policies may be computationally prohibitive. In this case, we recommend using the approximation methods that developed in Section 3.5. The alternating-optimization approach generates lower bounds on minimax regret that exhibit low error and may be used to guide query selection approaches that use information from the minimax regret computation. While exhibiting higher error, the reformulation-linearization technique (RLT) compliments alternating-optimization by constructing an upper bound that may be communicated to users as a guarantee on the current level of regret.

Future work that efficiently characterized the properties of IRMDPs that give rise to large sets of nondominated policies would complement the work in this chapter and offer guidance as to which domains are amenable to our methods for leveraging nondominated policies.

The investigation of structured MDP *dynamics* is somewhat orthogonal to the primary aims of this thesis; however, many of the algorithms detailed in this chapter could benefit from future work that explicitly models and leverages factored transition functions. Specifically, linear programming approaches (Guestrin, Koller, Parr, and Venkataraman, 2003a; Poupart et al., 2002) allow for very large factored MDPs to be compactly encoded and should dovetail nicely with the constrained optimization methods presented in this Chapter.

3.8.1 Contributions

Completed work from this section constitutes the following contributions to the literature:

- An exact approach to minimax regret computation using constraint generation and mixed integer programming (Regan and Boutilier, 2008, 2009)
- Two efficient approximate methods for efficient lower bounds on minimax regret (Regan and Boutilier, 2009)
- An efficient approximate method for upper bounds on minimax regret (Regan and Boutilier, 2008, 2011a)
- An exact approach to minimax regret computation leveraging nondominated policies (Regan and Boutilier, 2010)
- A polynomial algorithm for generating nondominated policies (Regan and Boutilier, 2010)

Chapter 4

Reward Elicitation

4.1 Introduction

Our minimax regret computation produces a robust policy with bounded regret that can be improved by reducing uncertainty over the reward function through elicitation. In this chapter we focus our attention on general methods for effectively *selecting* queries that make no assumptions about the structure of the reward function and restrict our attention to structurally simple query types. In the next chapter we explore how additive independence among attributes of a factored reward function may be exploited to construct highly targeted queries operating on small sets of attributes.

Let $\mathcal{Z}$ be the set of possible queries. For ease of notation we associate each query $Z \in \mathcal{Z}$ with a set of potential query responses, denoted $\rho \in \mathcal{P}(Z)$. Algorithm 4 summarizes our elicitation procedure. Note that elicitation can be viewed as a form of interactive optimization (Fisher, 1986) where significant interaction with a user occurs during the optimization process to further specify the model being optimized.

We begin by computing the solution $\langle \mathbf{f}, \mathbf{g}, \mathit{mmr} \rangle$ given the initial reward uncertainty $\mathcal{R}$. If the regret *mmr* is higher than the user's termination threshold τ, we select a query $Z \in \mathcal{Z}$. Each query response places additional constraints on $\mathcal{R}$, reducing the volume of $\mathcal{R}$ and potentially

Algorithm 4: Generic Reward Elicitation Procedure

```
Input:
    mdp:$\langle S, A, P, \gamma, \beta\rangle \leftarrow$ Underlying MDP
    $\mathcal{R} \leftarrow$ Initial imprecise reward specification specification
    $\tau \leftarrow$ Termination threshold
    $mmr \leftarrow$ Initial regret level ($\infty$)
Ouput:
    Recommended Policy $\mathbf{f}$, and final regret level $mmr$
$\langle \mathbf{f}, \mathbf{g}, mmr\rangle \leftarrow$ computeMMR(mdp,$\mathcal{R}$)
while $mmr > \tau$ do
    $Z \leftarrow$ selectQuery(mdp,$\mathcal{R}, \mathbf{f}, \mathbf{g}$)
    Administer query $Z$ and collect response $\rho$
    $\mathcal{R} \leftarrow \mathcal{R}^{Z\rightarrow\rho}$
    $\langle \mathbf{f}, \mathbf{g}, mmr\rangle \leftarrow$ computeMMR(mdp,$\mathcal{R}$)
end
```

lowering minimax regret. We denote $\mathcal{R}^{Z\rightarrow\rho}$ as the updated set of feasible reward functions given the additional constraints imposed by response $\rho \in \mathcal{P}(Z)$. Our goal is not to reduce reward uncertainty for its own sake; instead we wish to either lower minimax regret below some desired threshold τ, or to lower regret to zero, at which point the computed policy is provably optimal. The aim of query selection then is to directly and effectively reduce minimax regret.

While our policy selection criterion (i.e., minimax regret) assumes strict uncertainty, were there a precise quantification of the prior probability of all possible query responses, we could apply Bayesian methods to query selection—while preserving the regret guarantees of the recommended minimax regret optimal policy. To select the next query Z we could seek to minimize the *expected minimax regret* of the query polytope that results from the response.

$$\left[\text{Expected MMR}\right] \qquad Z^* \; = \; \operatorname*{argmin}_{Z\in\mathcal{Z}} \; \mathbb{E}^{\sigma}_{r\in\mathcal{R}}\left[\sum_{\rho\in\mathcal{P}(Z)} \Pr(\rho|Z,r) MMR(\mathcal{R}^{Z\rightarrow\rho})\right] \tag{4.1}$$

Note that this formulation assumes both a prior σ over feasible reward instantiations and a prior $\Pr(\rho|Z,r)$ over responses ρ given a query and reward function. The latter prior is known as the *response model* and it can be noisy or deterministic. In this work we focus on an approach that does not rely on the presence of fully specified prior distributions—instead we select the query that will guarantee the largest reduction of regret; i.e., the *worst-case regret* (WR) minimizing query.

$$[\text{WR}] \qquad Z^* = \operatorname*{argmin}_{Z \in \mathcal{Z}} \max_{\rho \in \mathcal{P}(Z)} MMR(\mathcal{R}^{Z \to \rho}) \tag{4.2}$$

We refer to Equation 4.2 as the *myopically optimal query*, since its purview is restricted to the immediate next step of elicitation. Considering the next k queries will more closely approximate the optimal *sequence* of queries (as $k \to \infty$), however, the optimization required will be intractable. Even when restricting look-ahead to $k = 1$, myopically optimal query selection (explored in Section 4.3) incurs substantial computational expense. We instead begin our discussion of query selection strategies with computationally efficient heuristics and proceed to myopically optimal query selection as a "gold standard" to demonstrate the effectiveness of our heuristics.

In conjunction with the query strategies we explore, there are many potential *types* of queries that may be used. The relevant aspects of a potential query type are: the cognitive burden it places on the user (i.e., the ease with which the user can form a response), the computational complexity of selecting its parameters, and its effectiveness with respect to the resulting reduction in minimax regret. There is often a tension between the conceptual simplicity of a query and the effectiveness of the query in reward elicitation. Queries which are constrained so as to be quickly understood lack the freedom of more complex queries to elicit the information that maximally reduces minimax regret.

From the users perspective, queries that involve the assessment and comparison of full policies are in many cases cognitively intractable, since they require the user to consider contingent actions for each possible state of the MDP. However, two recent approaches point toward more intuitive variants of full policy queries. The first explores techniques for explaining MDP policies to ordinary users (Khan, Poupart, and Black, 2009, 2011) using automatically generated templates that convey why each policy action has been chosen. We suggest a second approach in Section 7.5.5 that expresses a policy in terms of its impact on a small set of reward bearing states.

In Section 4.3 we suggest that while full-policy queries are complex, they may also be among the most informative queries, and are amenable to optimal selection of their query parameters using *setwise max regret*. For non-factored MDPs reward queries are far simpler for users to grasp, yet the user is still required to reason with respect to the entire state. Chapter 5 discusses how factored MDPs with additive independence in reward can allow for queries about a single attribute of the reward function in isolation. To contrast with the complex full policy queries used for myopically optimal query selection, we use simple bound queries in our discussion of heuristic query selection strategies. Experiments in Section 4.3.2 demarcate a spectrum of query types and strategies, with heuristic selection of bound queries at one end, and myopically optimal selection of full policy queries at the other end.

Other query types in this spectrum include the comparison of (full or partial) state-action trajectories or distributions over trajectories; and comparisons of outcomes in factored reward models. Both policy and trajectory comparisons can be facilitated by using counts of relevant (or reward-bearing) events as dictated by a factored reward model for example. The principles and heuristics detailed in this chapter can be adapted to these other query types.

4.2 Heuristic Query Selection

We focus on two approaches to heuristic query selection: 1) a *volumetric approach* that relies only on information about the uncertain reward polytope, and 2) a *current solution approach* which uses information generated by the minimax regret computation. We pair our simple heuristics with simple bound queries; however, our heuristic strategies can be adapted to more general query types. In the context of IRMDPs, a bound query takes the form *"Is* $r(s,a) \geq b$*?"* where b lies between the upper and lower bound on $r(s,a)$. While this appears to require a direct, quantitative assessment of value/reward by the user, it can be recast as a *standard gamble* (French, 1986), which can be constructed as follows. We assume that we know the reward for the most desirable state-action pair $(s^\top, a^\top)$ and least desirable state-action pair $(s^\perp, a^\perp)$. We define a standard gamble that asks the user to to choose whether they would prefer to realize the state-action pair (s, a) with certainty or the lottery $\langle(s^\top, a^\top), p, (s^\perp, a^\perp)\rangle$ where $p = [b - r(s^\perp, a^\perp)]/[r(s^\top, a^\top) - r(s^\perp, a^\perp)]$. A response to this standard gamble will yield the same linear constraint as a response to the original bound query. For simplicity, we express the query in bound form. Unlike reward queries (Delage and Mannor, 2007), which require an assessment of the exact value of $r(s, a)$, bound queries require only a yes-no response and are less cognitively demanding. A response tightens either the upper or lower bound.

There are many ways to select the point (s, a) at which to ask a bound query. We explore some simple heuristic criteria that are straightforward to compute building on work in (Boutilier et al., 2006).

4.2.1 Halve-the-Largest-Gap

Recall that the feasible reward set $\mathcal{R}$ is initially specified by a set of linear constraints $\mathbf{Cr} \leq \mathbf{d}$; $\mathcal{R}$ is closed under the the bound queries that we pose, since each bound query imposes linear constraints.

We define upper and lower bounds on the reward for each state-action pair as follows:

$$r^\top(s, a) = \max_{r \in \mathcal{R}} r(s, a) \qquad (4.3) \qquad\qquad r^\perp(s, a) = \min_{r \in \mathcal{R}} r(s, a) \qquad (4.4)$$

For general $\mathcal{R}$, the lower and upper bounds can be respectively computed by solving an LP for each of the $|S||A|$ state-action pairs (where each LPs is a direct formation of the optimization from Equation (4.3) or (4.4) respectively).

The upper bounds can quickly approximated by a single LP (4.5) that solves for all parameters at once.

$$r^{\top} = \underset{r \in \mathcal{R}}{\operatorname{argmax}} \sum_{s \in S} \sum_{a \in A} r(s,a) \qquad (4.5) \qquad\qquad r^{\perp} = \underset{r \in \mathcal{R}}{\operatorname{argmin}} \sum_{s \in S} \sum_{a \in A} r(s,a) \qquad (4.6)$$

The lower bounds can be similarly approximated using LP (4.6). When $\mathcal{R}$ forms a hyper-rectangle, the approximation is exact and (i.e., the respective bounds of (4.3) and (4.4) are exactly the respective bounds found by (4.6) and (4.6)). We use the upper and lower bound of each state-action pair to establish the *gap*, denoted $\Delta(s, a)$:

$$\Delta(s,a) = r^{\top}(s,a) - r^{\perp}(s,a) \tag{4.7}$$

Following Boutilier et al. (2006) we adapt the *halve largest gap* (HLG) heuristic to IRMDPs. The heuristic selects the point (s^*, a^*) with the largest gap,

$$[\text{HLG}] \qquad (s^*, a^*) = \underset{a \in A, s \in S}{\operatorname{argmax}} \; \Delta(s,a), \tag{4.8}$$

and queries the user about the midpoint $b = r^{\perp}(s^*, a^*) + \Delta(s^*, a^*)/2$ of that gap. Thus either response will reduce the interval by half. This process of directly reducing the volume of the reward polytope is motivated by theoretical considerations. In the context of single-step decision making, Boutilier et al. (2006) bound the resulting regret level (in terms of utility) after a fixed number of queries selected by HLG. In our sequential decision making context, the result implies a bound on our definition of minimax regret, in terms of policy value, after a fixed number of queries. The approach can further be viewed as a special-case of the polyhedral methods of Tobia et al. (2003a) operating on a hyper-rectangle. Intuitively, as reward uncertainty is reduced, minimax regret must eventually be reduced, producing an improved policy.

4.2.2 Current Solution Heuristics

The halve-the-largest gap heuristic can be augmented to focus on parameters that are directly involved in the solution to the current minimax regret computation. The *current solution* (CS) heuristic uses the occupancy frequencies from the minimax optimal solution f or the adversarial witness g to weight each gap. Intuitively, if a query involves a reward parameter that influences the value of neither f nor g (i.e., neither the player nor the adversary is forced to change their policy in the minimax regret computation), minimax regret may not be reduced, and visitation frequencies quantify the *degree* of influence. Formally CS selects the point:

$$[\text{CS}] \qquad (s^*, a^*) = \underset{a \in A, s \in S}{\operatorname{argmax}} \max\Big\{ f(s,a)\Delta(s,a),\ g(s,a)\Delta(s,a) \Big\}.$$

Given the selected (s^*, a^*), the query parameter b is set to the midpoint $r^{\perp}(s^*, a^*) + \Delta(s^*, a^*)/2$. The current solution heuristic does not admit the worst case regret reduction guarantees provided by HLG, however, the guidance of queries toward parameters that most impact regret significantly improves elicitation effectiveness in practice.

It is straightforward to apply CS to other robustness criteria such as the maximin value by using the visitation frequencies associated with the optimal policy w.r.t. to the chosen criterion.

4.2.3 Experiments

The proceeding experiments analyse the effectiveness of the HLG and CS heuristics for selecting queries. To reinforce our choice of the minimax regret criterion, we analyse a variant of our elicitation approach using an alternate robustness criterion: *maximin* (McMahan et al., 2003; Bagnell et al., 2003; Nilim and Ghaoui, 2005; Iyengar, 2005).

$$[\text{Maximin Value}] \qquad MM(\mathcal{R}) = \max_{\mathbf{f} \in \mathcal{F}} \min_{\mathbf{r} \in \mathcal{R}} \ \mathbf{f} \cdot \mathbf{r}$$

In the context of IRMDPs, maximin selects the policy that delivers the highest value given worst-case setting of reward. To compute the maximin optimal policy for IRMDPs, we implemented a variation of the algorithm developed by McMahan, Gordon and Blum (2003). The algorithm uses Benders' Decomposition (1962) to compute maximin value using a constraint generation procedure that mirrors the exact minimax regret computation technique discussed in Section 3.1. Constraint generation for maximin value uses following master and subproblem definitions:

$$\left[\text{master}\right] \qquad \begin{aligned} \underset{\delta,\mathbf{f}}{\text{maximize}} \quad & \delta \\ \text{subject to:} \quad & \delta \leq \mathbf{f}\cdot\mathbf{r} & \forall\, \mathbf{r} \in \mathit{GEN} \\ & \mathbf{E}^{\top}\mathbf{f} + \boldsymbol{\beta} = \mathbf{0}, \mathbf{f} \geq \mathbf{0} \end{aligned}$$

Where *GEN* is the restricted set of reward choices generated by the subproblem. Given a player choice of policy $\mathbf{f}$, the subproblem finds the wost possible setting of reward.

$$\left[\text{subproblem}\right] \qquad \begin{aligned} \text{minimize}\,\mathbf{r} \quad & \mathbf{f}\cdot\mathbf{r} \\ \text{subject to:} \quad & \mathbf{C}\mathbf{r} \leq \mathbf{d} \end{aligned}$$

The computation time for maximin is significantly less the that of minimax regret—this is expected since maximin requires only the solution of a pair of linear programs.

We use both maximin value and minimax regret to compute policies at each step of preference elicitation and pair each criterion with the current solution strategy, yielding (MM-CS) and (MMR-CS) respectively. We add a third strategy that uses halve-the-largest gap heuristic to select queries in conjunction with computing the minimax regret optimal policy; we denote this strategy (MMR-HLG). We assess each procedure by measuring the quality of the policy produced after each query, using the following metrics: (1) the maximin value given the policy and the current (remaining) reward uncertainty; (2) the max regret given the policy and the

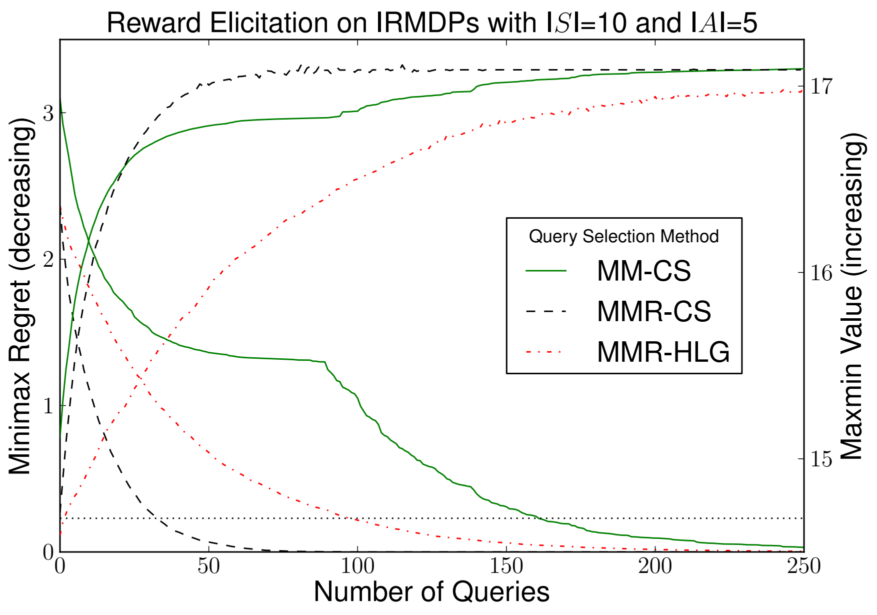


Figure 4.1: Reward Elicitation of various strategies measuring (decreasing) minimax regret on the left y-axis and (increasing) maximin value on the right y-axis. Results averaged over 100 randomly generated IRMDPs.

current (remaining) reward uncertainty. From our perspective max regret is the most critical since it provides the strongest guarantees.

Figure 4.1 shows results averaged over 100 IRMDPs, which were randomly generated using the procedure detailed in section 3.2. Each IRMDP has $|A| = 5$ actions and $|S| = 10$ states. MMR-CS naturally performs extremely well on the max regret measure. The dotted horizontal line in Figure 4.1 marks the point at which the MMR-CS strategy has reduced minimax regret to 10% of its original value; this reduction is achieved by MMR-CS in 32 queries; given the 50 reward parameters in this setting, this is significantly less than a single query per parameter. MMR-HLG takes more than three times that number of queries (97). For further comparison MM-CS takes more than five times the number of queries (161) to reduce regret to that level. MMR-CS reduces regret to zero after 97 queries; using less than 2 queries per reward parameter.

Turning our attention to the maximin measure, we see that while MMR-CS is not directly optimizing for maximin value, it is competitive with MM-CS, and after 20 queries it produces policies with better maximin value than the MM-CS procedure itself. This suggests that MMR-CS is selecting more informative queries, allowing for a larger reduction in reward uncertainty at the most relevant state-action pairs, leading to improvements in maximin value. This ability of MMR-CS to identify the highest impact reward points becomes clearer still when we examine the total reduction in the reward polytope over the course of elicitation.

Next we look at a measure of how reward uncertainty is reduced with each approach. Using volume to measure $\mathcal{R}$ does not distinguish between queries that halve a very small interval and queries that halve a large interval—the latter interval representing more reward uncertainty. Instead, we measure the sum of the length of the reward intervals using χ:

$$\chi = \sum_{s \in S} \sum_{a \in A} \Delta(s, a) \tag{4.9}$$

At the end of elicitation, MMR-HLG reduces χ to 5.9% of its original value (averaged over the 100 MDPs) and MM-CS reduces χ to 10.9% of its original value. MMR-CS only reduces χ to 67.8% of its original value—effectively eliminating regret while leaving a large amount of uncertainty in many of the reward parameters. Figure 4.2 illustrates this using a histogram of the number of queries asked by each method about each of the 5000 possible state-action pairs (50 pairs for each of the 100 runs). We see that MMR-CS asks zero queries about the majority (3422 out of 5000) of state-action pairs and asks a substantial number of queries (up to twelve) about a small number of "high impact" pairs.

The minimax regret computation used by the MMR-CS strategy took an average of 8.23 seconds over all queries used by all runs (for comparison, the maximin computation took an average of 0.16 seconds). An advantage of the HLG strategy is that does not require that minimax regret actually be computed. Minimax regret is only necessary to assess when to stop the elicitation process (i.e., to determine if minimax regret has dropped to an acceptable level). A possible modification the elicitation procedure to reduce time between queries is to adopt HLG and only compute minimax regret after every k queries. Of course, the HLG strategy will

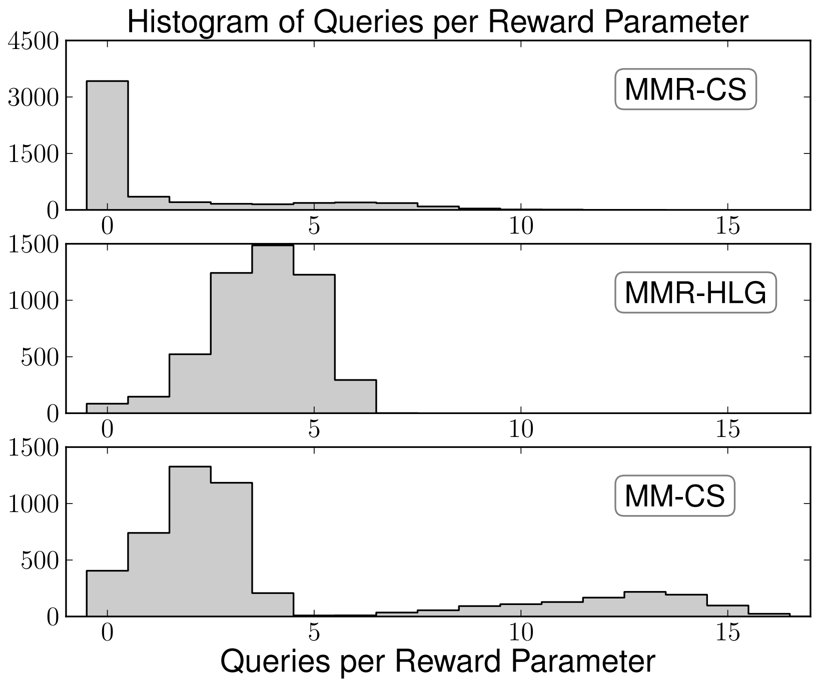


Figure 4.2: Histogram of number of queries at each state-action pair.

lead to a slower reduction in minimax regret as shown in Figure 4.1.

4.3 Myopically Optimal Query Selection

This section discusses how we can compute *myopically optimal policy queries* w.r.t. *worst-case regret (WR)*. Directly computing the query that minimizes WR is difficult due to interactions between the parameterization of the set feasible reward functions and the witness reward used to certify that the minimax regret optimal policy has been found. We adapt a result from the single-step elicitation literature that proves the equivalence of the WR-minimizing query to the query minimizing another measure—*set-wise max regret (SMR)*—which does not suffer from the same computational difficulties (Viappiani and Boutilier, 2009). This section formally defines WR and SMR and proves that for full-policy queries, the WR minimizing query is equal to the SMR minimizing query. We outline the details of the setwise max regret optimization using constraint generation and mixed integer programming and analyse the effectiveness of myopically optimal queries during elicitation.

We focus on *comparison queries*, requiring the user to state a preference between two policies. Define a query $Z = \{\mathbf{f}_a, \mathbf{f}_b\}$ as the two policies to be compared. Let $\mathcal{R}^{Z\to\mathbf{f}_i}$ denote the resulting set of feasible rewards consistent with the user's choice of $\mathbf{f}_i$ being the most preferred policy from the set Z:

$$\mathcal{R}^{Z\to\mathbf{f}_i} \equiv \left\{\mathbf{r} \in \mathcal{R} \mid \mathbf{f}_i\cdot\mathbf{r} \geq \mathbf{f}_j\cdot\mathbf{r} \quad \text{where} \quad \mathbf{f}_j \in Z \setminus \{\mathbf{f}_i\}\right\} \qquad i \in \{a, b\}$$

$$\text{e.g.} \quad \mathcal{R}^{Z\to\mathbf{f}_a} = \left\{\mathbf{r} \in \mathcal{R} \mid \mathbf{f}_a\cdot\mathbf{r} \geq \mathbf{f}_b\cdot\mathbf{r}\right\}$$

Given a user's choice of $\mathbf{f}_i$ from Z, the minimax regret of the resulting feasible reward set is $MMR(\mathcal{R}^{Z\to\mathbf{f}_i})$. We define the *worst case regret* (WR) of a query Z as:

$$\begin{aligned} WR(Z) &= \max_{\mathbf{f}_i\in\{\mathbf{f}_a,\mathbf{f}_b\}} \quad MMR(\mathcal{R}^{Z\to\mathbf{f}_i}) \\ &= \max\Big[\ MMR(\mathcal{R}^{Z\to\mathbf{f}_a}),\ MMR(\mathcal{R}^{Z\to\mathbf{f}_b})\ \Big] \end{aligned} \tag{4.10}$$

Let the set of potential queries be $\mathcal{Z} \equiv \{\ \{\mathbf{f}_a, \mathbf{f}_b\} \mid \mathbf{f}_a \in \mathcal{F}, \mathbf{f}_b \in \mathcal{F}\ \}$. We would like to compute the query $Z \in \mathcal{Z}$ that minimizes $WR(Z)$:

$$\min_{Z\in\mathcal{Z}} WR(Z) \quad = \quad \min_{Z\in\mathcal{Z}} \ \max_{\mathbf{f}_i\in Z} \ \min_{\mathbf{f}\in\mathcal{F}} \ \max_{\mathbf{r}\in\mathcal{R}^{Z\to\mathbf{f}_i}} \ \max_{\mathbf{f}'\in\mathcal{F}} \quad \mathbf{f}'\cdot\mathbf{r} - \mathbf{f}\cdot\mathbf{r} \tag{4.11}$$

Directly optimizing (4.11) is difficult since we are simultaneously varying the constraints defining the reward polytope and the adversary's choice of reward point $\mathbf{r}$ from within the polytope. In a related setting—recommending items given an uncertain utility function—Viappiani and Boutilier (2009) develop an approach to minimizing worst case regret by instead minimizing *setwise max regret* (SMR) . They prove equivalence between the query that minimizes SMR and the query that minimizes WR. We adapt this result to our setting; we begin by defining the setwise max regret w.r.t. a set $Z = \{\mathbf{f}_a, \mathbf{f}_b\}$ of policies:

$$\begin{aligned} SMR(Z) &\equiv \max\Big[\; MR(\mathbf{f}_a, \mathcal{R}^{Z\to\mathbf{f}_a}),\; MR(\mathbf{f}_b, \mathcal{R}^{Z\to\mathbf{f}_b}) \;\Big] \\ &= \max_{\mathbf{f}_i \in Z}\; MR(\mathbf{f}_i, \mathcal{R}^{Z\to\mathbf{f}_i}) \\ &= \max_{\mathbf{r}\in\mathcal{R}}\; \max_{\mathbf{f}'\in\mathcal{F}}\; \min_{\mathbf{f}_i\in Z} \quad \mathbf{f}'\cdot\mathbf{r} - \mathbf{f}_i\cdot\mathbf{r} \end{aligned} \tag{4.12}$$

Setwise max regret quantifies the maximal loss experienced by the user if we restrict the user to choosing a policy from the set Z.

Theorem 2 *The query $Z^* \in \mathcal{Z}$ that minimizes setwise max regret also minimizes worst-case regret. Formally,*

$$\textit{if} \quad Z^*_{smr} = \operatorname*{argmin}_{Z\in\mathcal{Z}} SMR(Z) \quad \textit{then} \quad WR(Z^*_{smr}) = \operatorname*{argmin}_{Z\in\mathcal{Z}} WR(Z).$$

Our proof of this theorem follows Viappiani and Boutilier (2009) and can be found in the Appendix A.3. This result allows us to shift our focus to SMR, which is more amenable to effective computation.

4.3.1 Setwise Max Regret Computation

The optimization for finding the query which minimizes setwise regret can be expressed as:

$$\min_{Z\in\mathcal{Z}} SMR(Z) \;=\; \min_{Z\in\mathcal{Z}}\; \max_{\mathbf{r}\in\mathcal{R}}\; \max_{\mathbf{f}'\in\mathcal{F}}\; \min_{\mathbf{f}_i\in Z} \quad \mathbf{f}'\cdot\mathbf{r} - \mathbf{f}_i\cdot\mathbf{r} \tag{4.13}$$

We solve this optimization with another variant of the constraint generation approach from Section 3.1 using a series of linear and mixed integer linear programs. To simply exposition we continue our assumption that the query is restricted to posing a choice between *two* policies:

$Z = \{\mathbf{f}_a, \mathbf{f}_b\}$. Equation (4.13) be be expressed as a single constrained minimization as follows:

$$\begin{aligned}
&\underset{\delta,\mathbf{f}_a,\mathbf{f}_b,I_\mathbf{r}}{\text{minimize}} && \delta \\
&\text{subject to:} && \delta \geq I_\mathbf{r}(\mathbf{f}'_\mathbf{r}\cdot\mathbf{r} - \mathbf{f}_a\cdot\mathbf{r}) && \forall\, \mathbf{r} \in \mathcal{R} \\
& && \delta \geq (1 - I_\mathbf{r})(\mathbf{f}'_\mathbf{r}\cdot\mathbf{r} - \mathbf{f}_b\cdot\mathbf{r}) && \forall\, \mathbf{r} \in \mathcal{R}
\end{aligned}$$

Here each $I_\mathbf{r}$ is a binary indicator variable indicating a choice of policy $\mathbf{f}_a$ from Z for each potential adversary reward $\mathbf{r}$ (the setting $I_\mathbf{r} = 0$ indicates the choice of $\mathbf{f}_b$); $\mathbf{f}'_\mathbf{r}$ denotes the optimal adversary policy w.r.t. each reward $\mathbf{r}$. Rather than formulating a constraint for each of the infinitely many reward points in $\mathcal{R}$, we generate only potentially active constraints. We use a constraint generation procedure with the master problem as follows:

$$\begin{aligned}
\text{MASTER} \quad &\underset{\delta,\mathbf{f}_a,\mathbf{f}_b,I_\mathbf{r}}{\text{minimize}} && \delta \\
&\text{subject to:} && \delta \geq I_\mathbf{r}(\mathbf{f}'_\mathbf{r}\cdot\mathbf{r} - \mathbf{f}_a\cdot\mathbf{r}) && \forall\, \langle\, \mathbf{r}, \mathbf{f}'_\mathbf{r}\, \rangle \in \mathit{GEN} \\
& && \delta \geq (1 - I_\mathbf{r})(\mathbf{f}'_\mathbf{r}\cdot\mathbf{r} - \mathbf{f}_b\cdot\mathbf{r}) && \forall\, \langle\, \mathbf{r}, \mathbf{f}'_\mathbf{r}\, \rangle \in \mathit{GEN}
\end{aligned} \tag{4.14}$$

We reformulate to remove the quadratic term ($I_\mathbf{r}^{\mathbf{f}_a}\cdot\mathbf{f}_a$) to form a mixed integer linear program.

$$\begin{aligned}
\text{MASTER} \quad &\underset{\delta,\mathbf{f}_a,\mathbf{f}_b,I_\mathbf{r}}{\text{minimize}} && \delta \\
&\text{subject to:} && \delta \geq \mathbf{f}'_\mathbf{r}\cdot\mathbf{r} - \mathbf{f}_a\cdot\mathbf{r} - (1 - I_\mathbf{r})\cdot M && \forall\, \langle\, \mathbf{r}, \mathbf{f}'_\mathbf{r}\, \rangle \in \mathit{GEN} \\
& && \delta \geq \mathbf{f}'_\mathbf{r}\cdot\mathbf{r} - \mathbf{f}_b\cdot\mathbf{r} - (I_\mathbf{r})\cdot M && \forall\, \langle\, \mathbf{r}, \mathbf{f}'_\mathbf{r}\, \rangle \in \mathit{GEN} \\
& && \gamma\mathbf{E}^\top\mathbf{f}_a + \boldsymbol{\beta} = 0, \;\; \mathbf{f}_a \geq \mathbf{0} \\
& && \gamma\mathbf{E}^\top\mathbf{f}_b + \boldsymbol{\beta} = 0, \;\; \mathbf{f}_b \geq \mathbf{0} \\
& && I_\mathbf{r} \in \{0, 1\} && \forall\, \mathbf{r} \in \mathit{GEN}
\end{aligned} \tag{4.15}$$

The constant M must be large enough to render the relevant constraints non-binding when the "opposite" policy is chosen. Note that, during constraint generation, as the set of generated constraints grows, so too will the number of indicator variables, since we use an indicator $I_{\mathbf{r}}$ for each reward $\mathbf{r}$ in the set of generated constraints.

The subproblem computes the reward point (and optimal policy at that reward point) which maximizes setwise regret given $\{\mathbf{f}_a, \mathbf{f}_b\}$ from the solution to the master problem:

SUBPROBLEM
$$\begin{aligned} &\underset{\delta,\, \mathbf{r}\in\mathcal{R},\, \mathbf{f}'\in\mathcal{F}}{\text{maximize}} && \delta \\ &\text{subject to:} && \delta \le \mathbf{f}'\!\cdot\mathbf{r} - \mathbf{f}_a\!\cdot\mathbf{r} \\ & && \delta \le \mathbf{f}'\!\cdot\mathbf{r} - \mathbf{f}_b\!\cdot\mathbf{r} \end{aligned} \tag{4.16}$$

We remove the quadratic term $\mathbf{f}'\!\cdot\mathbf{r}$ from the subproblem (as described in Equation (4.16)). Again we reformulate as a mixed integer program, adapting the MIP developed in Section 3.1 for finding max regret by explicitly representing the adversary's value and Q-value functions. Given the master solution $Z = \{\mathbf{f}_a, \mathbf{f}_b\}$, the formulation is as follows:

SUBPROBLEM
$$\underset{\delta,\mathbf{Q},\mathbf{V},\mathbf{I},\mathbf{r}}{\text{maximize}} \quad \delta \tag{4.17}$$

$$\begin{aligned} \text{subject to:} \quad & \delta \le \boldsymbol{\beta}\!\cdot\mathbf{V} - \mathbf{f}_a\!\cdot\mathbf{r} \\ & \delta \le \boldsymbol{\beta}\!\cdot\mathbf{V} - \mathbf{f}_b\!\cdot\mathbf{r} \\ & \mathbf{Q}_a = \mathbf{r}_a + \gamma\mathbf{P}_a\mathbf{V} && \forall\, a\in\mathcal{A} \end{aligned}$$

$$\mathbf{V} \le \mathbf{Q}_a + (1-\mathbf{I}_a)\!\cdot M \qquad \forall\, a\in\mathcal{A} \tag{4.18}$$

$$\mathbf{C}\mathbf{r} \le \mathbf{d} \tag{4.19}$$

$$\sum_a \mathbf{I}_a = 1 \tag{4.20}$$

$$\mathbf{I}_a(s) \in \{0,1\} \qquad \forall\, a\in\mathcal{A},\ s\in\mathcal{S} \tag{4.21}$$

Here $\mathbf{I}$ represents the adversary's policy, with $\mathbf{I}_a(s)$ denoting the probability of action a being taken at state s; constraints (4.20) and (4.21) restrict the policy to be deterministic. Constraint (4.18) ensures that the optimal value $V(s) = Q(s, a)$ for a single action a. Constraint (4.19) defines the space of feasible reward functions.

The precomputation techniques discussed in Section 3.6 can be leveraged to reduce the complexity of computing the subproblem. Given the set Γ of nondominated policies generated by πWitness (or the approaches to be discussed in Sections 6.2 and 6.3), we can compute the subproblem by solving a small linear program for each $\mathbf{g} \in \Gamma$:

SUBPROBLEM-ND

$$\begin{aligned} \underset{\delta,\, \mathbf{r}}{\text{maximize}} \quad & \delta && (4.22) \\ \text{subject to:} \quad & \delta \leq \mathbf{g}\cdot\mathbf{r} - \mathbf{f}_a\cdot\mathbf{r} \\ & \delta \leq \mathbf{g}\cdot\mathbf{r} - \mathbf{f}_b\cdot\mathbf{r} \\ & \mathbf{Cr} \leq \mathbf{d} \end{aligned}$$

The nondominated policy g with the largest objective value determines the maximally violated constraint. This optimization replaces MIP (4.17) with a series of linear programs. As with approaches to computing minimax regret using nondominated policies, the efficiency of LP (4.22) is directly linked with the number of nondominated policies.

4.3.2 Experiments

We assess the general performance of selecting optimal policy comparison queries using setwise max regret on randomly generated MDPs and compared the effectiveness of query selection to two alternative heuristic methods. We use randomly generated MDPs with a factored additive reward function[1]. We define the factored state $\mathrm{s} = \langle x_1, x_2, \ldots x_k \rangle$ with $k = 2, 3$, or 4 binary variables yielding $|\mathcal{S}| = 4, 8, 16$ respectively. The factored additive reward on k attributes is defined by: $r(s) = r(x_1) + \cdots + r(x_k)$. In each case we set the number of actions

[1]The use of factored MDPs for these experiments as a precursor to work using factored MDPs in Chapter 5.

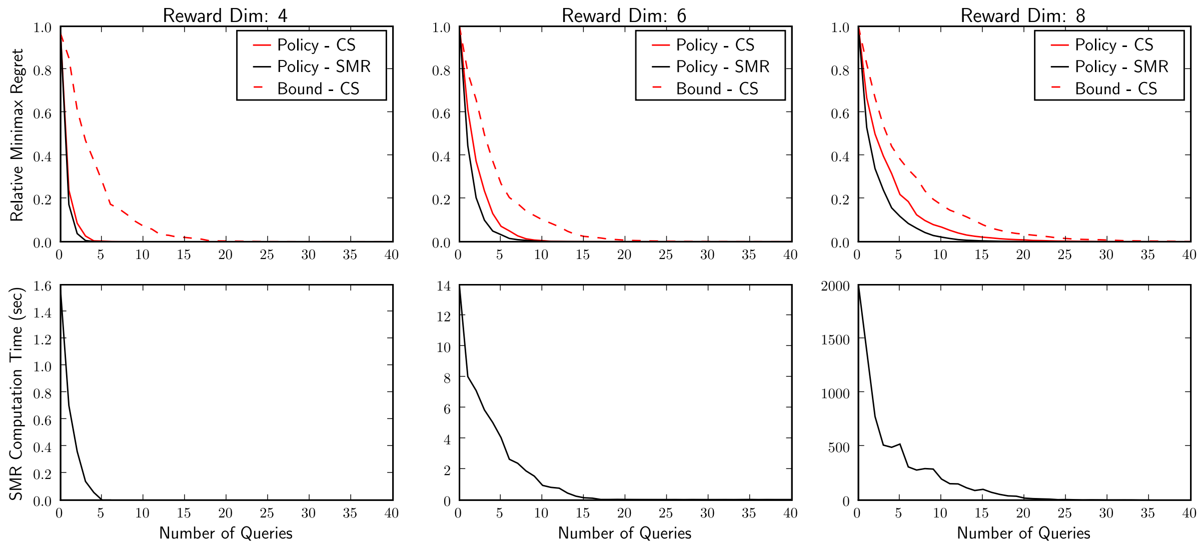


Figure 4.3: (top row) Relative minimax regret vs. query number during preference elicitation (averaged over 30 runs). (bottom row) Setwise max regret computation time (in seconds) for each query.

$|\mathcal{A}| = 5$. We impose a semi-sparse transition function as described in Section 3.2 and generate 30 MDPs of each size.

We compare our myopically optimal policy comparison query selection method (Policy-SMR) with bound queries chosen by the current solution heuristic from Section 4.2.2, denoted (Bound-CS). We examine one additional strategy that selects full policy comparison query $Z = \{\mathbf{f}, \mathbf{g}\}$, composed of the current solution (Policy-CS), where $\mathbf{g}$ is the adversarial policy that maximizes regret, and $\mathbf{f}$ is the minimax regret optimal policy.

Figure 4.3 shows the results (averaged over 30 runs) of preference elicitation using the three query strategies: Policy-SMR, Policy-CS, and Bound-CS. As expected, the myopically optimal queries selected by Policy-SMR outperformed the heuristic selection used by Policy-CS and Bound-CS. When $|\mathcal{R}| = 4$, Policy-CS is performing close to optimal (Fig 4.3. top-left plot). However as reward dimension increases, we can see that the performance gap between Policy-SMR and Policy-CS also increases (Figure 4.3 top-row). For $|\mathcal{R}| = 8$, regret is reduced to zero after 29 queries with Policy-CS, while Policy-SMR reduces regret to zero after only 18 queries. Policy queries impose a higher cognitive burden, however, they are also more informative relative to bound queries, allowing for fewer queries. Policy-SMR also requires the most computational overhead. Both heuristic selection methods use information that in most cases has already been computed (assuming the minimax regret is computed at each round of elicitation). The bottom row of Figure 4.3 shows SMR computation using MIP (4.17) to solve the subproblem (leveraging nondominated policies to solve the subproblem would likely further improve results). There are two important trends worth noting: first there is a significant (exponential) increase in computation time as reward dimension increases; second, the computation required by Policy-SMR decreases as elicitation progresses. Policy-SMR queries offer a principled approach to selecting queries that minimize the resulting regret of the worst case response. While our analysis suggests that Policy-SMR queries produce the quickest regret reduction during elicitation, its advantage over the other query strategies is not necessarily enough to justify the (significant) additional computation required. However, as elicitation progresses, it may eventually prove tractable to switch to myopically optimal query selection using SMR, further improving effectiveness.

4.4 Summary and Conclusions

Related work on inverse reinforcement learning provides an approach to specify reward functions through expert demonstration (Ng and Russell, 2000; Ratliff et al., 2006; Ramachandran and Amir, 2007; Coates et al., 2008; Ziebart et al., 2008); however, to the best of our knowledge the reward elicitation framework presented here is the first active, incremental approach to specifying reward functions: *active* in the sense that users are directly queried for their preferences, and *incremental* since, at each point during elicitation a robust policy with bounded regret is available to the user.

Considerable previous work has looked at regret-based elicitation that is both active and incremental (Patrascu, Boutilier, Das, Kephart, Tesauro, and Walsh, 2005; Boutilier et al., 2005, 2006; Viappiani and Boutilier, 2009; Braziunas and Boutilier, 2010; Boutilier, Regan, and Viappiani, 2010). However, the scope of this work is limited to single-step decision making contexts. We build on this work, extending the variants of HLG and CS heuristics developed to our sequential decision making domain (i.e., IRMDPs).

To complement these heuristic approaches for selecting simple bound queries, we describe a method for the optimal selection of more complex full policy queries. This optimal approach is inspired by work of the Viappiani and Boutilier (2009) that shows a more computationally tractable measure of setwise max regret (SMR) may be substituted for worst case regret (WR) when selecting queries (in single-step decision problems). We extended their result to address sequential decision problems contributing both: a theoretical component, proving that SMR can used to compute myopically optimal queries; and a practical component, detailing how the SMR optimizing query may be computed using constraint generation and mixed integer programming.

We undertook an empirical analysis focusing on the impact of optimal and heuristic query selection methods on reward elicitation. Of the heuristics we examined, using the current minimax regret optimal solution (MMR-CS) proved far more effective than using only volumetric information (MMR-HLG) or using the current solution w.r.t. other robustness criteria (MM-CS). The MMR-CS query strategy effectively steered elicitation toward high-impact reward parameters, quickly reducing regret (and increasing maximin value). We observed that the MMR-CS strategy identified provably optimal policies (i.e., reducing regret to zero) using a very small number of queries per reward parameter (less than a query per parameter on average).

Our experiments with myopically optimal selection of policy comparison queries demonstrated that elicitation effectiveness can be further improved. However, this approach imposes additional computational cost and cognitive burden. Chapter 6 will detail a domain in which this cognitive burden is lowered by expressing policy comparisons in terms of a small number of reward bearing states.

Experiments show that optimal policy comparison queries (Policy-SMR) outperform two alternative heuristic query methods, however the computation cost of selecting queries using setwise max regret grows quickly as reward dimension increases rendering optimal query selection using our setwise max regret technique infeasible for most realistic MDPs. To address computational difficulties, one avenue of future research is to investigate approximations to setwise max regret that lower the computational burden while selecting queries that remain more effective than current solution heuristics. The computational bottleneck lies in the subproblem of the constraint generation procedure which must compute $SMR(Z)$ for the solution Z found by the master problem. Equation (4.12) indicates that $SMR(Z)$ can be computed using a variant of max regret for each policy in the query Z (since $SMR(Z) = \max_{\mathbf{f}_i \in Z} MR(\mathbf{f}_i, \mathcal{R}^{Z \rightarrow \mathbf{f}_i})$). Re-expressing the subproblem in terms of two max regret computations allows us to leverage the many approximations that have been developed for the max regret computation. These approximations include alternating optimization, a relaxation of the original MIP formulation and a set of exact and approximate methods that make use of the set of nondominated policies.

Another interesting direction involves using setwise max regret to inform queries that have less cognitive burden than full-policy comparison queries. One possibility is selecting a single attribute over which to express a comparison query. For instance, given the min-SMR query: $Z = \{\mathbf{f}_a, \mathbf{f}_b\}$ and an attribute i: a query could ask the user to compare two lotteries over the instantiations of attribute i: "Would you prefer a policy in which attribute x_i occurred with probability $\mathbf{f}_a(x_i)$ for each $x_i \in X_i$ or would you prefer a policy in which x_i occurred with probability $\mathbf{f}_b(x_i)$ for each $x_i \in X_i$?". The occupancy frequency $\mathbf{f}(x_i)$ can be rounded to a whole number yielding "event counts" which may further simplify things for the user.

In this chapter we have developed a variety of approaches to reward elicitation in MDPs that eases the burden of reward function specification. The approach to minimax regret developed in Chapter 3 not only offers robust policies in the face of reward uncertainty, but it can be further leveraged to focus elicitation attention on the most important aspects of the reward function. While the computational costs are significant, it is an effective driver of elicitation, thus reducing the burden of reward determination.

4.4.1 Contributions

Completed work from this section constitutes the following contributions to the literature:

- The development of volumetric and current solution heuristics for IRMDPs (Regan and Boutilier, 2009)
- Proof of equivalence between min SMR and min WR for full policy queries (Regan, 2011)
- An exact approach for solving min SMR using constraint generation and mixed integer programming (Regan, 2011)

Chapter 5

Leveraging Reward Structure

Most sequential decision problems can be naturally expressed as factored MDPs (Boutilier et al., 1999) that use sets of state variables to compactly encode large state spaces. Typically reward depends only on a small fraction of these state variables and often in an additive way. This structure can help to streamline the elicitation of reward information.

In general user preferences over individual attributes are not naturally calibrated, thus we decompose reward into 1) *local reward functions* which capture user preferences over individual attributes, and 2) *scaling constants* that calibrate the contributions of the local reward. This decomposition is analogous to the definition of additive utility in terms of local *value* functions in Equation (2.2). In our additive model we develop decision-theoretically sound heuristics to simultaneously elicit information about local reward functions and scaling constants. We develop an exact algorithm for minimax regret computation based on our approach to flat reward models from Chapter 3, and address complications that arise due to parameter interactions between scaling constants and local reward in the reward representation. We describe how reward structure may be leveraged in example IRMDPs from the autonomic computing and assistive technology domains; we demonstrate substantial gains in elicitation effectiveness in these domains by applying our approach to structured reward.

5.1 Additive Reward with Local Reward Functions

We define the *factored* states of our IRMDP by a set of *attributes*: $\mathbf{X} = X_1 \times \ldots \times X_n$, where each X_i is an attribute with finite domain. For ease of notation we use X_i to refer to both the i^{th} attribute and its domain. Let $\mathbf{x} \in \mathbf{X}$ be a state from the set of possible states $\mathbf{X}$. We write $\mathbf{x}[i]$ to denote the value x_i of the i^{th} attribute of state $\mathbf{x}$. Let $\mathbf{x}_{\text{-}i}$ denote the restriction of the state $\mathbf{x}$ to the attributes excluding X_i; we write $(x_i, \mathbf{x}_{\text{-}i})$ to indicate the state obtained by conjoining the two. In addition, we assume a user's reward function is *additive independent* (Fishburn, 1967; Keeney and Raiffa, 1976) over the attribute space, conditional on the choice of action. Factored MDPs can also have factored actions, and admit compact specification of dynamics in addition to reward (Boutilier et al., 1999); however, we exploit only the factored nature of rewards, since our focus is on reward elicitation.

To formally characterize reward independence we adapt some notation from Chapter 2 on decision theory to factored MDPs. Let $\ell = \langle p_1, \mathbf{x}_1; \ldots; p_n, \mathbf{x}_n \rangle$ be a lottery over full states and let $\ell(\mathbf{x}_j)$ be the probability of the lottery realizing full state $\mathbf{x}_j = (x_1, \ldots, x_n)$. The marginal lottery $\ell_{\{i\}}$ over attribute X_i is defined by probabilities $\ell_{\{i\}}(x_i) = \sum_{\mathbf{x}_{\text{-}i} \in X_{\text{-}i}} \ell(x_i, \mathbf{x}_{\text{-}i})$.

Attributes $X_1, \ldots, X_n$ are additively independent iff $(\ell_{\{1\}}, \ldots, \ell_{\{n\}}) = (\ell'_{\{1\}}, \ldots \ell'_{\{n\}}) \Rightarrow \ell \sim \ell'$ (Fishburn, 1967). Thus, a reward function is additively decomposable if the user is indifferent between any lotteries ℓ and ℓ' over states $\mathbf{x} \in \mathbf{X}$ (given a fixed action a) whenever their marginals on each state attribute are the same; in this case we may express reward as:

$$r(\mathbf{x}, a) = \sum_i r_i(x_i, a) = \sum_i \lambda_i \upsilon_i(x_i, a), \tag{5.1}$$

where the r_i are *sub-reward functions* for each attribute, which are themselves expressed using *local utility functions* υ_i and *scaling constants* λ_i such that $r_i(x_i, a) = \lambda_i \upsilon_i(x_i, a)$. We assume explicit dependence on A for ease of exposition, but preferences for some/all attributes will be independent of A in many MDPs. If preferences over actions are independent of state, we can simply treat A as another attribute.

To aid concision we introduce the following vector notation: let $\boldsymbol{\lambda}$ be an n-vector with entries λ_i; and $\boldsymbol{\upsilon}$ an $n \times |\mathbf{X}||A|$ matrix with entries $\upsilon_i(\mathbf{x}[i], a)$. Below we write $\mathbf{f} \cdot \boldsymbol{\lambda}\boldsymbol{\upsilon}$ to mean $\mathbf{f} \cdot (\sum_i \lambda_i \boldsymbol{\upsilon}_i)$

We extend the "flat" reward elicitation techniques from Chapter 3 by incorporating regret-based approaches to *multi-attribute* decision problems (Boutilier et al., 2006; Braziunas and Boutilier, 2007). In the next section we describe some of the natural queries supported by the additive model. In Section 5.3 we give the details of how our approach to minimax regret computation from Chapter 3 may be extended to the handle additional complexities due to parameter interactions between scaling constants and local reward functions. We then develop a heuristic query selection strategy in Section 5.4 and demonstrate the effectiveness of our approach on example domains in Section 5.5.

5.2 Structured Query Types

The additive structure of the reward function admits both *local queries* involving only single attributes X_i and *global queries* that involve full state instantiations $\mathbf{x}$. Additive independence allows the local utility functions υ_i to be determined for each X_i independently. Global queries are only required to fix scaling constants λ_i, which calibrate strength of preference across attributes (Keeney and Raiffa, 1976; Fishburn, 1967).

Local Anchor Queries For any fixed action a, let $x_{i,a}^{\top}$ denote the most preferred value of attribute X_i given action a, and let $x_{i,a}^{\perp}$ denote the least preferred. *Local anchoring* is the process of asking a user to identify these values from the domain of X_i.

The numerical values for $\upsilon_i(x_{i,a}^{\top}, a)$ and $\upsilon_i(x_{i,a}^{\perp}, a)$ can be chosen arbitrarily, since local reward function, as we construct it, will be unique up to positive affine transformations (by the expected utility theorem and axioms described in Section 2.1.2). Given $\upsilon_i' = a\upsilon_i + b, a > 0$, both υ_i and υ_i' will represent the same preference relation over attribute X_i. To simply the expression of other queries w.r.t. the local anchors, we set $\upsilon_i(x_{i,a}^{\top}, a) = 1$ and $\upsilon_i(x_{i,a}^{\perp}, a) = 0$.

Local Bound Queries With local anchors in hand, we can determine $\upsilon_i(x_i, a)$ for any x_i using a *standard gamble query*: the user is asked to identify the probability p with which they would be indifferent between x_i and a *gamble* $\langle x^{\top}_{i,a}, p, x^{\perp}_{i,a}\rangle$ (which gives $x^{\top}_{i,a}$ with probability p, and $x^{\perp}_{i,a}$ with $1-p$). This determines $\upsilon_i(x_i, a) = p$.

Standard gambles, unfortunately, impose a significant cognitive burden on users due to the precision required. Instead we use *local bound queries*: given a constant b that we wish to use as a bound, we set the probability $p = [b - \upsilon_i(x^{\perp}_{i,a})]/[\upsilon_i(x^{\top}_{i,a}) - \upsilon_i(x^{\perp}_{i,a})]$. We ask the user whether x_i is preferred to the gamble $\langle x^{\top}_{i,a}, p, x^{\perp}_{i,a}\rangle$. A positive response implies $\upsilon_i(x_i, a) > b$, and a negative response that $\upsilon_i(x_i, a) \leq b$. One can ask such queries using gambles, or directly using the bound itself, depending on the context.

Global Anchoring Scaling parameters λ_i calibrate strength of preference across attributes; thus, they require global queries, but only of a specific form. *Global anchoring queries* ask the user to specify her most preferred and least preferred state-action pairs $(\mathbf{x}, a)^{\top}$ and $(\mathbf{x}, a)^{\perp}$, respectively. We fix a *reference outcome* $\mathbf{x}^0_a$ for $a \in A$. While not necessary, the same state can be chosen for each action a (reducing the number of full states a user is required to assess); if state rewards are independent of actions, dependence on a is not needed. The reference outcome can be any salient outcome; we set $\mathbf{x}^0_a = (x^{\perp}_{1,a}, \ldots, x^{\perp}_{n,a})$, which yields:

$$\begin{aligned} r((x^{\top}_{i,a}, \mathbf{x}^0_{\text{-}i}), a) &= r_i(x^{\top}_{i,a}, a) + \sum_{j \neq i} r_j(x^0_j, a) \\ &= r_i(x^{\top}_{i,a}, a) \end{aligned} \tag{5.2}$$

Global Bound Queries The reward $r((x^{\top}_{i,a}, \mathbf{x}^0_{\text{-}i}), a)$ can be elicited with a standard gamble asking for the probability p for which the user is indifferent between $((x^{\top}_{i,a}, \mathbf{x}^0_{\text{-}i}), a)$ and the lottery $\langle (\mathbf{x}, a)^{\top}, p, (\mathbf{x}, a)^{\perp}\rangle$. Given that $\upsilon_i(x^{\top}_{i,a}, a) = 1$ and $r_i(x^{\top}_{i,a}, a) = \lambda_i \upsilon_i(x^{\top}_{i,a}, a)$, Equation

5.2 implies that λ_i is exactly the reward elicited for outcome $r((x_{i,a}^{\top}, \mathbf{x}_{\text{-}i}^{0}), a)$:

$$\begin{aligned} r((x_{i,a}^{\top}, \mathbf{x}_{\text{-}i}^{0}), a) &= r_i(x_{i,a}^{\top}, a) && \big[\text{from (5.2)}\big] \\ &= \lambda_i \upsilon_i(x_{i,a}^{\top}, a) && \big[\text{from (5.1)}\big] \\ &= \lambda_i && \big[\text{since } \upsilon_i(x_{i,a}^{\top}, a) = 1\big] \end{aligned}$$

As with local queries, rather than asking precise numerical queries we use *global bound queries* which, given a bound b, derive the probability p (in the same manner as local bound queries) such that the response will imply a linear constraint in terms of b. The query asks which of $((x_{i,a}^{\top}, \mathbf{x}_{\text{-}i}^{0}), a)$ and $\langle (\mathbf{x}, a)^{\top}, p, (\mathbf{x}, a)^{\perp} \rangle$ is preferred. Notice global bound queries constrain each λ_i independently. The absence of constraints linking components of $\boldsymbol{\lambda}$ can be leveraged in minimax regret computation (discussed in Section 5.3).

To summarize, we elicit reward parameters by first asking the user to specify $x_{i,a}^{\top}$ and $x_{i,a}^{\perp}$ (local anchoring) for each X_i, a, and $(\mathbf{x}, a)^{\top}$ and $(\mathbf{x}, a)^{\perp}$ (global anchoring). We then use local bound queries to constrain local utilities $\boldsymbol{\upsilon}$ and global bound queries to constrain scaling factors $\boldsymbol{\lambda}$.

We have described a set of simple, focused queries that place constraints on the local reward functions υ_i and corresponding scaling constants λ_i. In order to fill in the picture of elicitation for our structured reward model, what remains is: 1) a method for computing policies that are robust to uncertainty over both $\boldsymbol{\lambda}$ and $\boldsymbol{\upsilon}$; and 2) procedures for selecting the parameters of queries to effectively reduce regret.

5.3 Computing Minimax Regret

In our additive reward model with local reward functions, we replace the set $\mathcal{R}$ specifying the uncertain parameters of the flat reward function with two sets: $\mathcal{R}_{\lambda}$ and $\mathcal{R}_{\upsilon}$ corresponding to the feasible scaling constants and feasible local reward functions respectively. We assume that these sets are defined by linear constraints using loose upper/lower bounds or the responses to any of the queries defined above,

$$\mathcal{R}_\lambda \equiv \{\boldsymbol{\lambda} \mid \mathbf{C}_\lambda \boldsymbol{\lambda} \leq \mathbf{d}_\lambda\}$$
$$\mathcal{R}_\upsilon \equiv \{\boldsymbol{\upsilon} \mid \mathbf{C}_\upsilon \boldsymbol{\upsilon} \leq \mathbf{d}_\upsilon\},$$

where $\mathbf{C}_\lambda$ and $\mathbf{C}_\upsilon$ are the matrices of coefficients, and $\mathbf{d}_\lambda$ and $\mathbf{d}_\upsilon$ are vectors of values representing these constraints. We extend our definitions of max regret and minimax regret to incorporate our new of parameterization of reward uncertainty:

$$\begin{aligned} PMR(\mathbf{f}, \mathbf{g}, \mathcal{R}_\lambda, \mathcal{R}_\upsilon) &= \max_{\boldsymbol{\lambda}\in\mathcal{R}_\lambda} \max_{\boldsymbol{\upsilon}\in\mathcal{R}_\upsilon} \; \mathbf{g}\cdot\boldsymbol{\lambda\upsilon} - \mathbf{f}\cdot\boldsymbol{\lambda\upsilon} && (5.3) \\ MR(\mathbf{f}, \mathcal{R}_\lambda, \mathcal{R}_\upsilon) &= \max_{\mathbf{g}\in\mathcal{F}} \max_{\boldsymbol{\lambda}\in\mathcal{R}_\lambda} \max_{\boldsymbol{\upsilon}\in\mathcal{R}_\upsilon} \; \mathbf{g}\cdot\boldsymbol{\lambda\upsilon} - \mathbf{f}\cdot\boldsymbol{\lambda\upsilon} && (5.4) \\ MMR(\mathcal{R}_\lambda, \mathcal{R}_\upsilon) &= \min_{\mathbf{f}\in\mathcal{F}} MR(\mathbf{f}, \mathcal{R}_\lambda, \mathcal{R}_\upsilon) && (5.5) \\ &= \min_{\mathbf{f}\in\mathcal{F}} \max_{\mathbf{g}\in\mathcal{F}} \max_{\boldsymbol{\lambda}\in\mathcal{R}_\lambda} \max_{\boldsymbol{\upsilon}\in\mathcal{R}_\upsilon} \; \mathbf{g}\cdot\boldsymbol{\lambda\upsilon} - \mathbf{f}\cdot\boldsymbol{\lambda\upsilon} && (5.6) \end{aligned}$$

Pairwise max regret $PMR(\mathbf{f}, \mathbf{g}, \mathcal{R}_\lambda, \mathcal{R}_\upsilon)$ is the maximal difference in value between policies $\mathbf{f}$ and $\mathbf{g}$ under possible reward realizations (parameterized by $\mathcal{R}_\lambda$ and $\mathcal{R}_\upsilon$). $MR(\mathbf{f}, \mathcal{R}_\lambda, \mathcal{R}_\upsilon)$ is the maximum regret of a policy $\mathbf{f}$ given the set of feasible scaling constants $\mathcal{R}_\lambda$ and the set of feasible local reward functions $\mathcal{R}_\upsilon$. Recall maximum regret represents the worst-case loss over possible realizations of reward—now parameterized by $\boldsymbol{\lambda\upsilon}$. $MMR(\mathcal{R}_\lambda, \mathcal{R}_\upsilon)$ is minimax regret which selects the policy that minimizes this loss.

To compute minimax regret we adapt our constraint generation approach from 3.1. At each iteration, two optimizations are solved. The *master problem* solves a relaxation of a program corresponding to Equation 5.6 that restricts the choices of g, $\boldsymbol{\lambda}$ and $\boldsymbol{\upsilon}$ by the adversary. Given the solution $\mathbf{f}$ to the master problem, the *subproblem* computes $MR(\mathbf{f}, \mathcal{R}_\lambda, \mathcal{R}_\upsilon)$ and adds the solution $(\mathrm{g}, \boldsymbol{\lambda}, \boldsymbol{\upsilon})$—capturing the most violated constraint from the master problem—to the set of choices available to the adversary in the master problem. When the max regret computed by the subproblem equals the minimax regret computed by the master problem then we have converged to the exact solution to minimax regret. The following LP formulates the master problem in our structured reward context:

$$\begin{aligned}
\underset{\mathbf{f},\delta}{\text{minimize}} \quad & \delta && (5.7)\\
\text{subject to:} \quad & \delta \geq \boldsymbol{\beta}\cdot\mathbf{V}_i - \mathbf{f}\cdot\boldsymbol{\lambda}_i\boldsymbol{\upsilon}_i && \forall\ \langle \mathbf{V}_i, \boldsymbol{\lambda}_i, \boldsymbol{\upsilon}_i\rangle \in \mathit{GEN}\\
& \gamma\mathbf{E}^\top\mathbf{f} + \boldsymbol{\beta} = \mathbf{0}\\
& \mathbf{f} \geq \mathbf{0}
\end{aligned}$$

This formulation is essentially our original master problem formulation (3.19) with a different reward parameterization. GEN is a set of constraints corresponding to a subset of possible adversarial choices of policies and rewards; $\boldsymbol{\beta}\cdot\mathbf{V}_i = \mathbf{g}_i\cdot\boldsymbol{\lambda}_i\boldsymbol{\upsilon}_i$, the value of the adversary's policy corresponding to constraint i. If GEN contains all vertices of the polytopes $\mathcal{R}_\lambda, \mathcal{R}_\upsilon$ and the corresponding optimal value function $\mathbf{V}$, this LP computes minimax regret exactly. However, most constraints will not be active at optimality, so iterative constraint generation is used: given solution $\mathbf{f}$ to the relaxed problem with a subset GEN of constraints, we solve the *subproblem* finding the most violated constraint, i.e., the $\boldsymbol{\lambda}, \boldsymbol{\upsilon}, \mathbf{V}$ that maximizes regret of $\mathbf{f}$. If no violated constraints exist, then $\mathbf{f}$ is optimal. Our construction of the subproblem formulation for generating violated constraints begins with minimal changes to the original subproblem formulation for flat IRMDPs. To aid with our comparison between new and original formulations we repeat the original here:

$$\begin{aligned}
\underset{\mathbf{Q},\mathbf{V},\mathbf{I},\mathbf{r}}{\text{maximize}} \quad & \boldsymbol{\beta}\cdot\mathbf{V} - \mathbf{f}\cdot\mathbf{r} && && (3.14)\\
\text{subject to:} \quad & \mathbf{Q}_a = \mathbf{r}_a + \gamma\mathbf{P}_a\mathbf{V} && \forall\, a \in A\\
& \mathbf{V} \geq \mathbf{Q}_a && \forall\, a \in A\\
& \mathbf{V} \leq (1-\mathbf{I}_a)\mathbf{M}_a + \mathbf{Q}_a && \forall\, a \in A\\
& \mathbf{C}\mathbf{r} \leq \mathbf{d}\\
& \sum_a \mathbf{I}_a = \mathbf{1}\\
& \mathbf{I}_a(s) \in \{0,1\} && \forall a \in A, s \in S\\
& \mathbf{M}_a = \mathbf{M}^\top - \mathbf{M}_a^\perp
\end{aligned}$$

The indicator vectors $\mathbf{I_a}$ represent the adversary's policy, with $I_a(s)$ denoting the probability of action a being taken at state s. Constraints (3.17) and (3.18) restrict the policy to be deterministic. Constraints (3.15) and (3.16) ensure that the optimal value $V(s)$ will be set to the Q-Value $Q(s, a)$ for at a single action a. The vector $\mathbf{M}_a$ of "big-M" constants renders constraints non-binding (see Chapter 3). Recall that (3.14) is a mixed integer *linear* program. To adapt the subproblem to our structured reward parameterization we replace $\mathcal{R}$ with $\mathcal{R}_\lambda$ and $\mathcal{R}_\upsilon$:

$$\begin{aligned}
\underset{\mathbf{Q},\mathbf{V},\mathbf{I},\boldsymbol{\lambda},\boldsymbol{\upsilon}}{\text{maximize}} \quad & \boldsymbol{\beta}\cdot\mathbf{V} - \mathbf{f}\cdot\boldsymbol{\lambda}\boldsymbol{\upsilon} && && (5.8)\\
\text{subject to:} \quad & \mathbf{Q}_a = \boldsymbol{\lambda}\boldsymbol{\upsilon}_a + \gamma\mathbf{P}_a\mathbf{V} && \forall\, a \in A\\
& \mathbf{V} \geq \mathbf{Q}_a && \forall\, a \in A\\
& \mathbf{V} \leq (1-\mathbf{I}_a)\mathbf{M}_a + \mathbf{Q}_a && \forall\, a \in A\\
& \mathbf{C}_\lambda\boldsymbol{\lambda} \leq \mathbf{d}_\lambda\\
& \mathbf{C}_\upsilon\boldsymbol{\upsilon} \leq \mathbf{d}_\upsilon\\
& \sum_a \mathbf{I}_a = \mathbf{1}\\
& \mathbf{I_a}(\mathbf{x}) \in \{\mathbf{0},\mathbf{1}\} && \forall\, a \in A, \mathbf{x} \in \mathbf{X}\\
& \mathbf{M}_a = \mathbf{M}^\top - \mathbf{M}_a^\perp
\end{aligned}$$

The term $\mathbf{f} \cdot \mathbf{r}$ from (3.14) is replaced by $\mathbf{f} \cdot \boldsymbol{\lambda} \boldsymbol{\upsilon}$. The constraint on reward $\mathbf{Cr} \le \mathbf{d}$ from (3.14) is replaced by the constraints $\mathbf{C}_\lambda \boldsymbol{\lambda} \le \mathbf{d}_\lambda$ and $\mathbf{C}_\upsilon \boldsymbol{\upsilon} \le \mathbf{d}_\upsilon$ on scaling constants and local reward respectively. Recall that $\mathbf{I}_a$ is an $|S|$-vector of indicator variables denoting whether (adversarial) policy g takes action a at the corresponding state; and $\mathbf{M}$ is a vector of sufficiently large constants.

Our new formulation (5.8) contains the quadratic term $\boldsymbol{\lambda}\boldsymbol{\upsilon}$. However, when no constraints link scaling constants $\boldsymbol{\lambda}$ (e.g., as with global bound queries), an optimal solution must set λ_i to its maximum $\lambda_i^\top$ or minimum $\lambda_i^\perp$. In such a case, we can linearize the MIP as follows. We introduce indicators $\mathbf{J}$, where $J_i = 1$ means $\lambda_i = \lambda_i^\top$ and $J_i = 0$ means $\lambda_i = \lambda_i^\perp$. In the optimal solution we have: $\boldsymbol{\lambda}\boldsymbol{\upsilon} = \mathbf{J}\boldsymbol{\lambda}^\top \boldsymbol{\upsilon} - (1 - \mathbf{J})\boldsymbol{\lambda}^\perp \boldsymbol{\upsilon}$. The quadratic term $\mathbf{J}\boldsymbol{\upsilon}$ is linearized using a "big-M" formulation in which we introduce a continuous variable $\mathbf{Z}$ which replaces $\mathbf{J}\boldsymbol{\upsilon}$, with constraints $\mathbf{Z} \le \boldsymbol{\upsilon} + (\mathbf{1} - \mathbf{J})\mathrm{M}'$ and $\mathbf{Z} \le \mathbf{J}\mathrm{M}'$. The constant M' is set sufficiently large so as to invalidate the constraint when present (i.e., when not multiplied by zero). The result is a (linear) MIP with continuous variables $\mathbf{Q}, \mathbf{V}, \mathbf{Z}, \boldsymbol{\lambda}, \boldsymbol{\upsilon}$ and binary variables $\mathbf{I}, \mathbf{J}$:

$$\begin{aligned}
\underset{\mathbf{Q},\mathbf{V},\mathbf{I},\mathbf{J},\mathbf{Z},\boldsymbol{\upsilon}}{\text{maximize}} \quad & \boldsymbol{\beta}\mathbf{V} - \mathbf{f}\Big[\mathbf{Z}\boldsymbol{\lambda}^\top - \boldsymbol{\lambda}^\perp \boldsymbol{\upsilon} + \mathbf{Z}\boldsymbol{\lambda}^\perp\Big] && && (5.9)\\
\text{subject to:} \quad & \mathbf{Q}_a = \Big[\mathbf{Z}_a\boldsymbol{\lambda}^\top - \boldsymbol{\lambda}^\perp \upsilon_a + \mathbf{Z}_a\boldsymbol{\lambda}^\perp\Big] + \gamma\mathbf{P}_a\mathbf{V} && \forall\, a \in A \\
& \mathbf{V} \ge \mathbf{Q_a} && \forall\, a \in A \\
& \mathbf{V} \le (\mathbf{1} - \mathbf{I_a})\mathbf{M_a} + \mathbf{Q_a} && \forall\, a \in A \\
& \mathbf{M_a} = \mathbf{M}^\top - \mathbf{M}_\mathbf{a}^\perp && \forall\, a \in A \\
& \mathbf{Z} \le \boldsymbol{\upsilon} + (\mathbf{1} - \mathbf{J})\mathbf{M}' && && (5.10)\\
& \mathbf{Z} \le \mathbf{J}\mathbf{M}' && && (5.11)\\
& \mathbf{C}_\lambda \boldsymbol{\lambda} \le \mathbf{d}_\lambda \\
& \mathbf{C}_\upsilon \boldsymbol{\upsilon} \le \mathbf{d}_\upsilon \\
& \sum_a \mathbf{I_a} = \mathbf{1} \\
& \mathbf{I_a}(\mathbf{x}) \in \{\mathbf{0}, \mathbf{1}\} && \forall\, a \in A, \mathbf{x} \in \mathbf{X} \\
& J_i \in \{0, 1\} && \forall\, i = 1, \ldots, n && (5.12)
\end{aligned}$$

The term $\boldsymbol{\lambda v}$ has been linearized using the new variables $\mathbf{J}$ and $\mathbf{Z}$, and constraints (5.10) to (5.12) have been added to realize this linearization. This formulation allows us to generate constraints (solving max regret) using a mixed integer *linear* program with $O(n|\mathbf{X}||A|)$ continuous variables, $O(n|\mathbf{X}||A|)$ binary variables and $O(n|\mathbf{X}||A|)$ constraints. Exact minimax regret computation remains a significant bottleneck that prevents solving any but the smallest MDPs, leading us to develop approximations, to which we now turn.

5.3.1 Approximate Minimax Regret

While constraint generation with MIP (5.9) solves minimax regret exactly, exact solutions are not needed to effectively guide query selection (Boutilier et al., 2006): approximations often suffice. Furthermore, the linearization of the quadratic MIP for $MR(\mathbf{f}, \mathcal{R})$ only works when independent upper bounds are available on the λ_i. To this end, we extend the alternating approximation approach developed in Section 3.5 to construct a tractable approach to minimax computation.

Alternating Approximation

We repeatedly compute, in turn, an optimal (adversarial) policy g (fixing $\boldsymbol{\lambda}, \boldsymbol{v}$), scaling factors λ (fixing g, $\boldsymbol{v}$), and local utility functions $\boldsymbol{v}$ (fixing g, $\boldsymbol{\lambda}$). This reduces the max regret computation to the following sequence of LPs:

Given $\mathbf{f}, \boldsymbol{\lambda}, \boldsymbol{v}$:

$$\begin{aligned} \underset{\mathbf{g}}{\text{maximize}} \quad & \mathbf{g}\cdot\boldsymbol{\lambda v} - \mathbf{f}\cdot\boldsymbol{\lambda v} \\ \text{subject to:} \quad & \gamma\mathbf{E}^{\top}\mathbf{g} + \boldsymbol{\beta} = \mathbf{0},\ \mathbf{g} \geq \mathbf{0} \end{aligned}$$

Given $\mathbf{f}, \mathbf{g}, \boldsymbol{\lambda}$:

$$\begin{aligned} \underset{\boldsymbol{v}}{\text{maximize}} \quad & \mathbf{g}\cdot\boldsymbol{\lambda}\boldsymbol{v} - \mathbf{f}\cdot\boldsymbol{\lambda}\boldsymbol{v} \\ \text{subject to:} \quad & \mathbf{C}_v\boldsymbol{v} \leq \mathbf{d}_v \end{aligned}$$

Given $\mathbf{f}, \mathbf{g}, \boldsymbol{v}$:

$$\begin{aligned} \underset{\boldsymbol{\lambda}}{\text{maximize}} \quad & \mathbf{g}\cdot\boldsymbol{\lambda}\boldsymbol{v} - \mathbf{f}\cdot\boldsymbol{\lambda}\boldsymbol{v} \\ \text{subject to:} \quad & \mathbf{C}_\lambda\boldsymbol{\lambda} \leq \mathbf{d}_\lambda \end{aligned}$$

This approach locally explores solution space, avoiding the potentially expensive solution of the linear MIP; and it is applicable even when the MIP cannot be linearized. Alternating optimization is guaranteed to converge and must return a feasible solution (w.r.t. the subproblem), and quality can be further improved using random restarts. Finally, when used in constraint generation, it provides a lower bound on minimax regret.

Reformulation-Linearization Technique

We also apply the reformulation-linearization technique (RLT) introduced in Section 3.5 to efficiently generate upper bounds on minimax regret. Given the max regret computation as a single (cubic) maximization:

$$\begin{aligned} \underset{\mathbf{g},\boldsymbol{\lambda},\boldsymbol{v}}{\text{maximize}} \quad & \mathbf{g}\cdot\boldsymbol{\lambda}\boldsymbol{v} - \mathbf{f}\cdot\boldsymbol{\lambda}\boldsymbol{v} \\ \text{subject to:} \quad & \gamma\mathbf{E}^\top\mathbf{g} + \boldsymbol{\beta} = \mathbf{0},\ \mathbf{g} \geq \mathbf{0} \\ & \mathbf{C}_\lambda\boldsymbol{\lambda} \leq \mathbf{d}_\lambda \\ & \mathbf{C}_v\boldsymbol{v} \leq \mathbf{d}_v \end{aligned} \tag{5.13}$$

RLT transforms the problem by introducing a set of variable factors using the problem constraints. Combinations of these factors are multiplied with the original problem constraints to generate additional valid nonlinear constraints. The resulting nonlinear program is then lin-

$\mathcal{R}$ dim.		Runtime (secs)		MMR			
$\|\boldsymbol{\lambda}\|$	$\|\boldsymbol{\upsilon}\|$	MIP	Alt.	MIP	Alt.	Error (Abs.)	Error (Rel.)
2	4	0.51	0.40	90.66	88.79	1.87	2.0 %
3	6	5.65	0.77	107.94	107.45	0.49	0.5 %
4	8	1744.20	1.09	149.52	149.19	0.33	0.2 %

Table 5.1: Performance of Alternative Approximation

earized by replacing each nonlinear term with a new variable. The solution to the resulting linear program (detailed in Appendix C.2) provides an upper bound on the optimal value to the original problem.

5.3.2 Assessment

We assessed the performance of exact minimax regret computation with the alternating approximation (using 10 random restarts) on a small set of random IRMDPs with 2–4 binary attributes generated using the approach from 3.2. Results are averaged over 30 runs and shown Table 5.1. While runtime of the exact formulation grows exponentially with reward dimension, approximation runtime and error remain low enough to admit real-time computation during interactive elicitation. This suggests that alternating approximation will provide an effective substitute for the MIP w.r.t. guiding elicitation.

The alternating model gives only a lower bound on MMR. Upper bounds are also valuable, since they offer a guarantee on max regret at any point during elicitation—allowing for termination when this guarantee meets the user's threshold τ.

Next, we assessed the effectiveness of RLT by computing an upper bound on MMR with the IRMDPs above, with results in the Table 5.2. The linearization is, naturally, much more efficient than the MIP, but produces weak upper bounds, that on average are an order of magnitude larger than the exact solution. However, Section 5.5 demonstrates that linearization often gives better approximations when: (i) the MDP is structured; and (ii) $\mathcal{R}_\lambda$ and $\mathcal{R}_\upsilon$ are sufficiently constrained.

$\mathcal{R}$ dim.		Runtime (secs)		MMR			
$\lvert\boldsymbol{\lambda}\rvert$	$\lvert\boldsymbol{\upsilon}\rvert$	MIP	Lin.	MIP	Lin.	Error (Abs.)	Error (Rel.)
2	4	0.51	0.18	90.66	1019.28	928.61	1024 %
3	6	5.65	0.31	107.94	1564.87	1456.93	1349 %
4	8	1744.20	0.63	149.52	2236.93	2087.41	1396 %

Table 5.2: Performance of Reformulation-Linearization Approximation

5.4 Query Selection

We limit our discussion of query selection to heuristic approaches. The additional complexities of minimax regret computation in our additive model render myopically optimal query selection computationally prohibitive, and inadvisable, especially given the demonstrated effectiveness of the heuristics we develop in this section.

At each round of elicitation we select a query, whose response ρ refines our knowledge of $\mathcal{R}_\lambda$ or $\mathcal{R}_\upsilon$, producing $\mathcal{R}_\lambda^{Z\to\rho}$ or $\mathcal{R}_\upsilon^{Z\to\rho}$ respectively; ideally reducing minimax regret. To select suitable queries, we adapt the *current solution* (CS) heuristic developed in Section 4.2, which uses the solution to the minimax optimization for query selection. Let $(\mathbf{f}^c, \mathbf{g}^c, \boldsymbol{\lambda}^c, \boldsymbol{\upsilon}^c)$ be the solution given current reward polytopes $(\mathcal{R}_\lambda, \mathcal{R}_\upsilon)$: $\mathbf{f}^c$ is the minimax optimal policy; $\boldsymbol{\lambda}^c, \boldsymbol{\upsilon}^c$ are the reward parameters that maximize regret of $\mathbf{f}^c$; and $\mathbf{g}^c$ is adversarial policy (which will be optimal for $\boldsymbol{\lambda}^c, \boldsymbol{\upsilon}^c$).

To inform selection among different query types (e.g., global bound vs. local bound) we introduce a score $S(\phi)$ for the parameters ϕ of a query that measures the potential to reduce the max regret of $\mathbf{f}^c$. We develop scoring functions for each query type below. The score can be combined with other factors relevant to query selection, e.g., the cognitive effort required to answer a query. However, we leave investigations of improved scoring functions to future work. Our adapted CS heuristic selects the query with the highest score.

For defining scores below we use the concept of occupancy frequencies w.r.t. specific attribute instantiation. Formally, let $f_i(x_i, a)$ be the *marginal occupancy frequencies* for an instantiation of state-attribute and action (x_i, a) given occupancy frequencies f:

$$f_i(x_i, a) = \sum_{\mathbf{x}_{\text{-}i} \in \mathbf{X}_{\text{-}i}} f((x_i, \mathbf{x}_{\text{-}i}), a)$$

The marginal occupancy frequency $f_i(x_i, a)$ corresponds to the discounted probability of being in a state with attribute x_i and taking action a under occupancy frequencies f.

5.4.1 Local Bound Scoring

A local bound query requires selection of an attribute-value, action pair (x_i, a) and local utility bound p. Define the *local utility gap* of (x_i, a) to be:

$$\upsilon\text{-}gap(x_i, a) = \uparrow\upsilon_i(x_i, a) - \downarrow\upsilon_i(x_i, a),$$

where $\uparrow \upsilon_i(x_i, a)$ and $\downarrow \upsilon_i(x_i, a)$ are the maximum and minimum values $\upsilon_i(x_i, a)$ can take in $\mathcal{R}_\upsilon$, when fixing $\upsilon^c_j(x_j, a)$ for all $j \neq i$. In general, the maximum and minimum values can be respectively computed by solving an LP for each attribute-value pair. When the reward parameters for each attribute-value action pair are independent (e.g., $\mathcal{R}$ is hyper-rectangular) the upper and lower bounds can be respectively computed by a single LP that solves for all parameters at once.

We fix the bound p in the local bound query for (x_i, a) at the midpoint of this gap (hence any response narrows the gap by half). To score (x_i, a), notice that

$$PMR(\mathbf{f}, \mathbf{g}, \mathcal{R}_\lambda, \mathcal{R}_\upsilon) = \max_{\boldsymbol{\lambda}, \boldsymbol{\upsilon}} \sum_i \lambda_i \sum_{x_i} \sum_a \Big[g_i(x_i, a) - f_i(x_i, a) \Big] \upsilon_i(x_i, a). \tag{5.14}$$

Thus, the impact on PMR of tightening the bound on $\upsilon_i(x_i, a)$ is mediated by the difference in policy marginals $g_i(x_i, a) - f_i(x_i, a)$ and the scaling constant λ_i. Assuming that we fix the query bound p to be the gap midpoint, any response to the query gives a new constraint on

$\upsilon_i(x_i, a)$ that changes PMR by at most

$$S_\upsilon(x_i, a) \equiv \lambda_i^c \Big[g_i(x_i, a) - f_i(x_i, a)\Big] \upsilon\text{-}gap(x_i, a) \Big/ 2. \tag{5.15}$$

Thus Equation (5.15) is our score for the parameters $\phi = (x_i, a)$ of local bound queries.

5.4.2 Global Bound Scoring

A global bound query requires selection of an attribute i and a bound p. Similar to local bound queries, define the *scaling gap*

$$\lambda\text{-}gap(i) = \uparrow\lambda_i - \downarrow\lambda_i,$$

where $\uparrow\lambda_i$ and $\downarrow\lambda_i$ are the max/min values that λ_i can take in $\mathcal{R}_\lambda$ given other reward parameters fixed by $\boldsymbol{\lambda}^c, \boldsymbol{\upsilon}^c$. In general, the maximum and minimum values can be respectively computed by solving an n LPs, one for each attribute i. When the scaling constants are independent, the upper and lower bounds can be respectively computed by a single LP that solves for all parameters at once.

We query the midpoint of this gap, a response will induce a constraint such that λ_i changes by at most $\lambda\text{-}gap(i)/2$. We similarly define the score of an anchor bound query parameters $\phi = (i)$ to measure its potential for reducing regret:

$$S_\lambda(i) \equiv \sum_{x_i \in X_i} \sum_{a \in A} \Big[g_i(x_i, a) - f_i(x_i, a)\Big] \upsilon_i^c(x_i, a) \lambda\text{-}gap(i) \Big/ 2$$

5.5 Experiments

We now examine two of our example domains that are naturally modeled using IRMDPs with additive reward, and explore the effectiveness of our approach to reward elicitation.

5.5.1 Assistive Technology

We present a simplified model of the COACH system (Boger et al., 2005), whose general goal is to guide a patient with dementia through a task with ℓ steps, such as hand-washing, using verbal or visual cues, while minimizing intrusion. Prompts can be issued at increasing levels of intrusiveness until (at the highest level k) a caregiver is called to assist the person in task completion. This results in action space $A = \{0, 1, \ldots, k\}$. The state is defined by three variables $\mathcal{S} = \langle T, D, F\rangle$; $T \equiv \{0, 1, \ldots, \ell\}$ is the number of tasks steps successfully completed by the person, $D \equiv \{0, 1, 2, 3, 4, 5+\}$ is the *delay* (time taken during the current step), and $F \equiv \{0, 1, \ldots, k\text{-}1\}$ tracks whether a prompt at a specific level was attempted on the current task step, but failed to immediately get the person to the next step. The dynamics express the following intuitions. The no-prompt action will cause a "progress" transition to the next step (setting delay and failed-prompt to zero), or a "stall" transition (same step with delay increased by one). The probability of reaching the next step with action $a = n$ is higher than $a = n-1$ since more intrusive prompts have a better chance of facilitating progress; however, progress probability decreases as delay increases. Reaching the next step after prompting is less likely if a prompt has already failed at the current step. The reward function is:

$$r(t, d, f, a) = r_g(t) + r_d(d) + r_p(a),$$

where: $r_g(t)$ is a positive "task completion" reward (non-zero if $t = \ell$ task, zero otherwise); $r_d(d)$ is a negative "delay" penalty; and $r_p(a)$ is a negative "prompting" penalty associated with prompting the person. Typically, $r_p(a = k)$ is a very large negative cost for calling the caregiver (relative to other costs); and the sub-reward functions $r_d(d)$ and $r_p(a)$ are both assumed to be monotonic (in delay and prompting level, respectively). Each sub-reward function is the product of a scaling constant and local utility function: $r_i = \lambda_i v_i$. Attribute F does not occur in the reward function, so requires no elicitation. A full specification for this domain can be found in Appendix E.1.

Local Anchoring

Local anchoring requires no elicitation from the user, since the best/worst values for each attribute T, D, A are given by assumption: penalties for D and A increase monotonically in level, while $t = \ell$ is known to be the preferred value of T (while $\upsilon_g(t) = 0$ for all $t \neq \ell$). Since we specify local utility functions on a $[0, 1]$ scale, we use negative scaling constants to ensure that the sub-reward functions for "penalties" are negative.

Global Anchoring

Our domain also permits global anchoring without input from the user. The most preferred state-action pair occurs when the task has been completed with no delay and no prompting: $(\mathbf{x}, a)^{\top} \equiv (t = \ell, d = 0, a = 0)$. The least preferred pair is when no progress is made beyond the first step despite the most intrusive prompts: $(\mathbf{x}, a)^{\perp} \equiv (t = 0, d = 5+, a = k)$. We set $r(\mathbf{x}_{\top}, a_{\top}) = 1$ and $r(\mathbf{x}_{\perp}, a_{\perp}) = -1$. In this case $r(\mathbf{x}^{\perp}, a^{\perp}) \neq 0$, and we set reference state $(\mathbf{x}^0, a^0) \equiv (t = 0, d = 0, a = 0\ )$.

Local Bound

Local bound queries in this domain are reasonably natural, especially for professional caregivers for whom our potential elicitation application is designed. For instance, consider a query involving υ_d, with this structure: $d = 3 \succeq \langle d = 0, b, d = 5+\rangle$? For bound $b = 0.6$, we might pose this as: "Would you prefer a certain delay of 3 minutes or a situation in which there is a 60% chance of delay 0 and a 40% chance of delay 5+? (all else being equal)". Local bound queries involving υ_p are specified similarly. Since only $\upsilon_g(\ell)$ has positive reward, no local elicitation of υ_g is needed.

Global Bound

We calibrate the scaling constants $\boldsymbol{\lambda}$ using global bound queries. For instance, consider a query involving λ_d of the form: $(d^{\perp}, \mathbf{x}^0_{\text{-}d}, a^0) \succeq \langle (\mathbf{x}, a)^{\top}, b, (\mathbf{x}, a)^{\perp} \rangle$. For $b = 0.2$ this query could be expressed as: "Would you prefer the situation in which at step one there is no delay, and a prompt of level k was issued (calling the caregiver); or would you prefer the following: 20% of the time the goal is reached with no delay and no prompting, but 80% of the time progress is not made beyond the first step and the maximal delay occurs despite the most intrusive prompting?" Given our anchoring assumptions, a positive response constraints $\lambda_d \leq 1 - 2b$.[1] We can bound λ_p in a similar fashion. Note that $\lambda_g = 1$ is implied by our other settings.

Assessment

We assessed the effectiveness of elicitation on an instance of this domain with $\ell = 10$ steps and $k = 4$ prompt levels. We simulate elicitation using a reward function reflecting the actual use of the COACH system (Boger et al., 2005) to generate user responses. The reward function has 12 (unknown) parameters and the size of the state-action space is $|\mathbf{X}||\mathbf{A}| = 960$. This is large enough that exact minimax regret computation is not fast enough to support real-time interaction with a user. Instead we use the alternating approximation to compute minimax regret; this provides us with a lower bound and max regret. We used RLT to compute an upper bound as well. We use the current solution strategy to select local and global bound queries. To determine the advantage of explicitly modeling the additive reward structure in the COACH domain, we also performed elicitation on the same IRMDP using a flat reward model in the reward function has $|\mathbf{X}||\mathbf{A}| = 960$ dimensions. In this case we use the query selection strategy and the approximate minimax algorithm (which again gives a lower bound on MMR) introduced in Section 3.5. This can be viewed as the flat model variant of our additive approach. Results—shown in Figure 5.1—are averaged over 30 runs with randomized initial $(\mathcal{R}_\lambda, \mathcal{R}_\upsilon)$.

A comparison of lower bounds shows a clear advantage for additive elicitation. Comparing the *upper* bound for the additive approach with the *lower* bound for the flat model, we see the additive model yields significant advantage, provably reducing minimax regret to zero after

[1]This constraint is different than if all υ_i were positive, but follows directly from $r(d^\perp, \mathbf{x}^0_{\text{-}d}, a^0) \geq r(\langle(\mathbf{x}, a)^\top, b, (\mathbf{x}, a)^\perp\rangle)$.

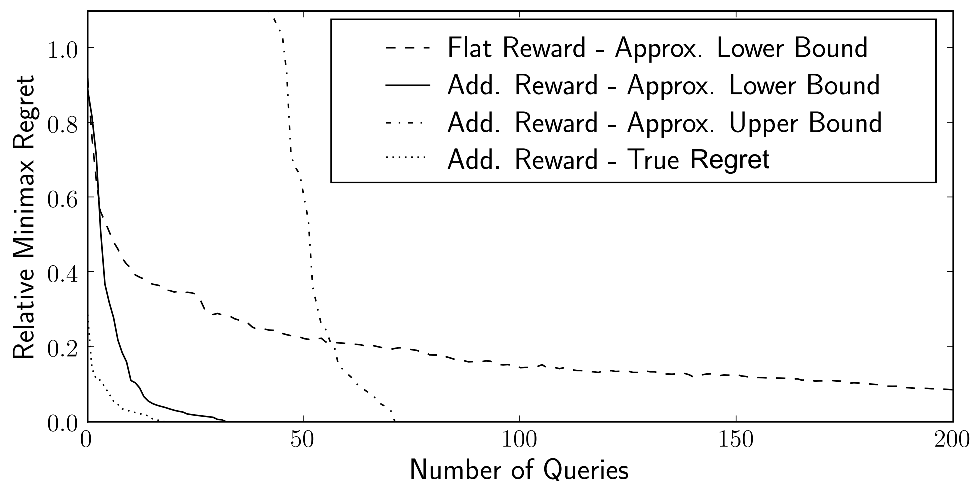


Figure 5.1: Preference Elicitation in COACH Domain (30 runs)

70 queries (the lower bound estimates MMR=0 after roughly 28 queries). By comparison, after 200 queries 8% of the original regret remains in the *lower bound* produced by the flat reward model. Furthermore, queries for the flat reward function involve comparisons not over individual attribute values, but of full state-action instantiations, placing additional cognitive burden on the user. We also plot *true regret*, capturing the actual regret (loss) of the minimax optimal policy in the additive model w.r.t. to the true (but hidden) reward function. We see that true regret is significantly less than the lower bound on minimax regret. This suggests that (earlier) termination using the lower bound on max regret would be beneficial in practice.

5.5.2 Autonomic Computing

The autonomic computing task (Kephart and Chess, 2003) involves allocating computing or storage resources to servers as computing demands from clients change over time. Generally, server utility has no closed form: utility for a *specific* allocation is bounded using a combination of simulation and numerical optimization. Precise specification of utility, while automated, can involve significant cost.

We have *K application server elements* $e_1 \ldots e_k$, and N *units of resource* which may be assigned to the server elements (plus a "zero resource"). An allocation is specified by $\mathbf{n} =$

$\langle n_1 \ldots n_k \rangle$ where $\sum_i^K n_i \leq N$. Finally there are D *demand levels* at which each server element can operate. A full specification of demand levels is denoted $\mathbf{d} = \langle d_1 \ldots d_k \rangle$.

A state is given by the current allocation of resources and current demand levels for each server: $\mathbf{x} = (\mathbf{n}, \mathbf{d})$. Actions are allocations $\mathbf{m} = \langle m_1 \ldots m_k \rangle$ of up to N units of resource to the K application servers. Uncertainty in demand is exogenous and the action in the current state uniquely determines the allocation in the next state. Thus the transition function is composed of i Markov chains $\Pr(d_i' \mid d_i)$, one for the demand at each server element.

The reward $r(\mathbf{n}, \mathbf{d}, \mathbf{m}) = u(\mathbf{n}, \mathbf{d}) - c(\mathbf{n}, \mathbf{d}, \mathbf{m})$ is composed of a positive utility $u(\mathbf{n}, \mathbf{d})$ and the negative cost $c(\mathbf{n}, \mathbf{d}, \mathbf{m})$. The cost $c(\mathbf{n}, \mathbf{d}, \mathbf{m})$ is the sum of the costs of taking away one unit of resource from each server element at each time step. We assume that the cost term is known. The utility term $u(\mathbf{n}, \mathbf{d})$ can be factored into local utility functions $v_i(n_i, d_i)$ for each server i. In this setting, utility functions are defined with respect to a common unit (potential revenue), so there is no need for calibration: $\boldsymbol{\lambda} = \mathbf{1}$. A full specification for this domain can be found in Appendix E.2.

Reward elicitation proceeds by having a server bound its local utility for a given demand and resource level. An example query is: “Given n units of resource and a demand level of d, are the potential earnings generated greater or equal to b”. An answer of “yes” imposes the following constraint on the local utility function: $v_i(n_i, d_i) \geq b$.

Assessment

We examined the effectiveness of our approach to elicitation of additive reward in this domain using a small IRMDP with $K = 2, N = 3, D = 3$, yielding a reward function of dimension 24. We used the current solution heuristic to select queries and approximated minimax regret using alternating optimization (and RLT to produce an upper bound). As with COACH, we compare to elicitation using a flat version of the reward model in the same IRMDP (with 90 states/reward parameters) using the same methodology as above. Results are shown in Figure 5.2. We see once again that elicitation proceeds quickly and that taking advantage of the

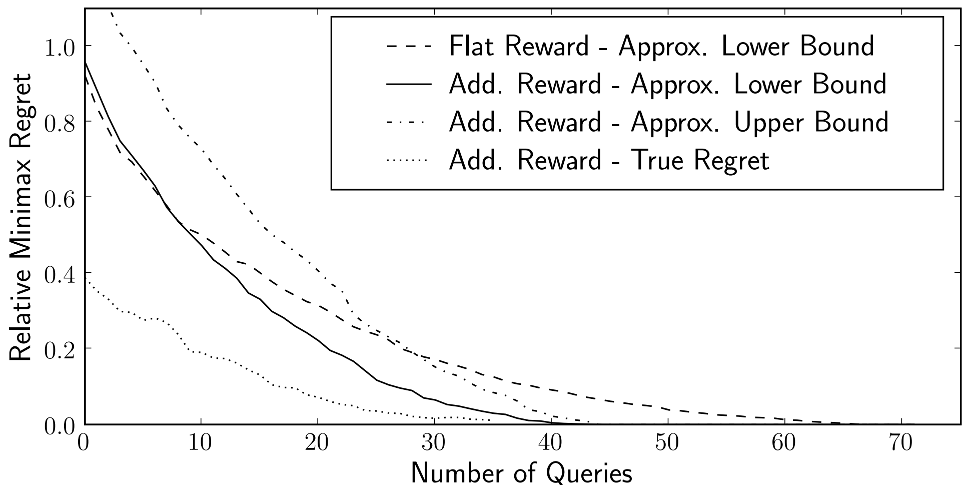


Figure 5.2: Preference Elicitation in Autonomic Domain (30 runs)

additive independence in the reward model yields considerable leverage. The upper bound on max regret generated by the additive model is reduced to zero after 44 queries, guaranteeing that an optimal policy has been found. For comparison, the flat model takes 68 queries for the *lower bound* to reach zero; thus it takes *at least* 24 more queries for an optimal policy to be elicited for the flat model (and likely it would take significantly more queries to reduce an exact measure of regret to zero).

5.6 Summary and Conclusions

Related work has developed approaches to elicitation of structured utility functions in single-step decision making domains (Gonzales and Perny, 2004; Braziunas and Boutilier, 2007, 2008, 2010). This work assumes general additive utility (GAI) and thus encompasses a broader set of possible utility functions than the strictly additive model we adopt. We extend the many of the query types and selection heuristics of Boutilier and Braziunas (2007) to our sequential decision making domain. This enables reward elicitation for IRMDPs whose reward functions exhibit additive independence. As in one-shot multi-attribute decision problems (Boutilier et al., 2005), this structure admits more cognitively manageable, yet decision-theoretically sound, queries involving single attributes, and requires very few global queries.

We developed an exact approach to computing minimax regret optimal policies for IR-MDPs with additive reward and local reward functions that allows for guarantees on loss to be provided during elicitation. In the general case, an exact method uses a series of linear and mixed integer quadratic programs; we identify a special (though common) case in which the mixed integer program may be linearized. We provided tools for efficient approximation of minimax regret in this context by extending the alternating optimization and the relaxation-linearization technique (RLT) to handle additive reward structure with local reward functions. The accuracy of alternating approximation makes it a suitable substitute for generating the current solution in the CS query selection heuristic, allowing for efficient yet effective query selection. RLT while far less accurate, provides a valid upper bound on regret that may be communicated to the user.

To demonstrate the effectiveness of our approach to eliciting additive rewards, we undertook an experimental assessment in two example domains. In each domain we observed that leveraging reward structure led to far more effective elicitation. In comparison, variants of each domain with unstructured reward: 1) took more queries and 2) required more demanding full-state queries. Our experimental assessments focused primarily on the computational aspects reward elicitation and carry certain limitations: The local queries that we propose are theoretically simple, however, we have not provided evidence that this simplicity translates into true cognitive ease on the part of real users. A third example domain detailed in Chapter 7 provides further argument toward cognitively manageable queries; however these arguments also lack data from user experiments. Future work that performs a user study on reward elicitation for a real-world domain is needed to establish the link between the theoretical simplicity of our queries and the ease with which real users can understand and respond to each query.

Though our approximations address the computational difficulty of minimax regret to an extent, further research would be beneficial. One promising approach is enlisting the set of nondominated policies developed in Section 3.6 (and in next chapter). This requires extending the definition of nondominance to cover policies that are simultaneously nondominated w.r.t. $\mathcal{R}_\lambda$ and $\mathcal{R}_\upsilon$, while addressing the computational complexities that arise from the interaction between $\boldsymbol{\lambda}$ and $\boldsymbol{\upsilon}$ in the computational machinery that generates and uses the set of nondominated policies.

As with our algorithms for flat IRMDPs, the minimax regret algorithms detailed in this chapter could benefit from future work that explicitly models and leverages factored dynamics (transition functions). Related work on compactly encoding linear programs to solve extremely large factored MDPs (Guestrin et al., 2003a; Poupart et al., 2002) should be directly translatable to our constrained optimization approach to minimax regret computation.

Our additive assumption, can be relaxed to allow *generalized additive independence (GAI)* (Fishburn, 1967). Techniques for elicitation and computing minimax regret in GAI models (Braziunas and Boutilier, 2007) should be readily adaptable to MDPs.

In this chapter we developed an approach to reward elicitation for MDPs with additive reward functions composed of local reward functions. We described a simple set of queries that restrict attention to single attributes leading to significantly more effective elicitation. In the next chapter we shift focus back to minimax regret computation in general (flat) IRMDPs. We describe how the set of nondominated policies may be intelligently wielded during elicitation to quickly and accurately approximate minimax regret.

5.6.1 Contributions

Completed work from this section constitutes the following contributions to the literature:

- An exact approach to computing MMR for IRMDPs with additive reward structure using constraint generation and mixed integer programming (Regan and Boutilier, 2010, 2011a)
- Two approximate approaches that provide both upper and lower bounds on MMR
- Decision-theoretically sounds heuristics for eliciting additive reward using local reward functions (Regan and Boutilier, 2011a)

Chapter 6

Online Minimax Regret Computation

During elicitation, the feasible reward set shrinks as more information is gleaned about the user's reward. Policies that were nondominated w.r.t. reward may become dominated when the feasible reward set is reduced. Since the computational performance of our approach to computing minimax regret using the set of nondominated policies is tightly tied to its cardinality, pruning away newly dominated policies can offer a significant speed up in computation.

While pruning can improve the efficiency of online computation, it also creates space to add new nondominated policies to the approximate set. Thus we can improve the quality of the approximation by adding new policies while maintaining the same online computational overhead by keeping the number of nondominated policies roughly constant through the effective use of pruning. This section details an algorithm for online adjustment of the nondominated policy set and demonstrates the effectiveness of the approach for elicitation. Figure 6.1 diagrams how this online adjustment fits into the overall reward elicitation framework.

6.1 Online Adjustment of Nondominated Policies

Let $\tilde{\Gamma} \subset \Gamma$ be a strict subset of the set of all nondominated policies Γ; we begin by describing how we can adjust this set online to compute increasingly accurate approximations of minimax regret during reward elicitation. During elicitation, the feasible reward set $\mathcal{R}$ shrinks as more

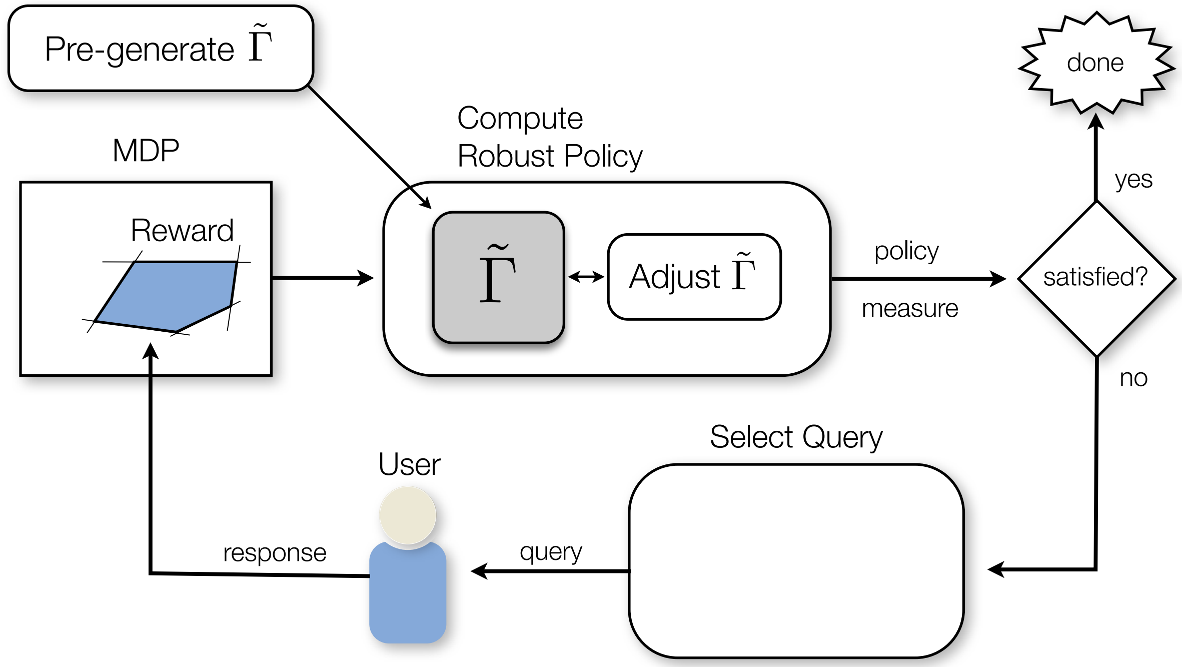


Figure 6.1: Diagram of reward elicitation framework with online adjustment of $\tilde{\Gamma}$, the approximate set of nondominated policies.

information is gleaned about the actual reward (e.g., as users respond to queries or behavior is observed). If $\mathcal{R}' \subset \mathcal{R}$ is the *refinement* of $\mathcal{R}$ implied by this additional information, then $\Gamma(\mathcal{R}') \subseteq \Gamma(\mathcal{R})$; policies that were nondominated w.r.t. $\mathcal{R}$ may become dominated when the feasible reward set is reduced to $\mathcal{R}'$. It is also the case that policies in $\tilde{\Gamma} \subset \Gamma(\mathcal{R})$ may be dominated w.r.t. the refinement $\mathcal{R}' \subset \mathcal{R}$—these dominated policies can be eliminated without increasing approximation error (w.r.t. minimax regret). In Section 3.6.1 we observed that the computational performance of constraint generation using a set of nondominated policies is tightly tied to the cardinality of the set, hence pruning away newly dominated policies can offer tremendous speed up in minimax regret computation.

Let $\tilde{\Gamma}$ be a (not necessarily complete) set of policies that are nondominated w.r.t. to $\mathcal{R}$ and let $\mathcal{R}' \subset \mathcal{R}$ be a refinement of $\mathcal{R}$. The pruning of $\tilde{\Gamma}$ w.r.t. to the new reward set $\mathcal{R}'$ can be realized as follows: for each policy $\mathbf{f} \in \tilde{\Gamma}$, we solve a simple LP to find a reward point at which

$\mathbf{f}$ is nondominated:

$$\begin{aligned} \underset{\mathbf{r},\delta}{\text{maximize}} \quad & \delta \\ \text{subj. to:} \quad & \delta \leq \mathbf{f}\cdot\mathbf{r} - \mathbf{f}'\cdot\mathbf{r}, \qquad\qquad \forall\ \mathbf{f}' \in \tilde{\Gamma} \setminus \mathbf{f} \\ & \mathbf{r} \in \mathcal{R}' \end{aligned} \tag{6.1}$$

If the objective δ is negative, then there is no reward $\mathbf{r} \in \mathcal{R}'$ for which $\mathbf{f}$ offers improvement over some $\mathbf{f}' \in \tilde{\Gamma}$. Formally: $\nexists\ \mathbf{r} \in \mathcal{R}'$: $\mathbf{f}\cdot\mathbf{r} \geq \mathbf{f}'\cdot\mathbf{r},\ \forall\ \mathbf{f}' \in \tilde{\Gamma}$; thus $\mathbf{f}$ is dominated and can be pruned from $\tilde{\Gamma}$. While pruning can speed up online computation, it can also be viewed as "creating space" to add new nondominated policies to the approximate set $\tilde{\Gamma}$. We can improve the quality of the approximation by adding new nondominated policies to $\tilde{\Gamma}$, while maintaining the same online computational overhead by keeping the size of $\tilde{\Gamma}$ roughly constant through the effective use of pruning. Adding new policies can be accomplished by running further iterations of a nondominated policy generation algorithm that allows for anytime computation. We develop such an algorithm in the Section 6.2. An alternative to anytime generation is to compute the entire set Γ offline and selectively add policies as elicitation proceeds; however, as Figures 3.8 and 3.9 from Chapter 3 demonstrate, nondominated policy generation can take significantly longer than computing minimax regret using the generated set. It follows that IRMDPs for which it is feasible to fully compute Γ offline will often admit efficient minimax regret computation without resorting to approximations of Γ.

We focus on the case where prior to online optimization, we have computed an initial set of nondominated policies $\tilde{\Gamma}_0 \subseteq \Gamma(\mathcal{R}_0)$ given the the initial feasible reward set $\mathcal{R}_0$. The size of $\tilde{\Gamma}_0$ is determined by the demands of efficient online minimax regret computation. At iteration t, minimax regret is computed for $\mathcal{R}_{t\text{-}1}$ using $\tilde{\Gamma}_{t\text{-}1}$. The new set $\mathcal{R}_t$ is formed (incorporating constraints given by a query response), and the set $\tilde{\Gamma}_t$ is constructed by: (a) pruning policies in $\tilde{\Gamma}_{t\text{-}1}$ that are dominated relative to $\mathcal{R}_t$; and (b) adding new nondominated policies to $\tilde{\Gamma}_{t\text{-}1}$ to improve approximation quality. Algorithm 5 outlines how pruning and addition can be integrated into an elicitation algorithm.

Algorithm 5: Online Adjustment during Elicitation

```
Input:
    $mdp$ : $\langle S, A, P, \gamma, \beta \rangle \leftarrow$ the parameters of our MDP model
    $\mathcal{R}_0 \leftarrow$ initial reward polytope
    $\tilde{\Gamma}_0 \leftarrow$ initial nondominated policies (computed offline)
    $\tau \leftarrow$ acceptable level of regret
Ouput:
    Recommended Policy $\mathbf{f}$, and final regret level *mmr*
$\langle \mathbf{f}, \mathrm{g}, mmr \rangle \leftarrow$ ComputeMMR(mdp,$\mathcal{R}_{t\text{-}1}, \tilde{\Gamma}_{t\text{-}1}$)
foreach step $t \geq 1$ do
    $\langle \mathbf{f}, \mathrm{g}, mmr \rangle \leftarrow$ ComputeMMR(mdp,$\mathcal{R}_{t\text{-}1}, \tilde{\Gamma}_{t\text{-}1}$)
    $Z \leftarrow$ SelectQuery($\mathbf{f}$, g,mdp)
    Administer query $Z$ and collect response $\rho$
    $\mathcal{R}_t \leftarrow \mathcal{R}_{t\text{-}1}^{Z \rightarrow \rho}$
    $\tilde{\Gamma}_t \leftarrow$ Prune($\mathcal{R}_t, \tilde{\Gamma}_{t\text{-}1}$) $\cup$ Add($\mathcal{R}_t, \tilde{\Gamma}_{t\text{-}1}$)
    if $mmr < \tau$ then
        terminate and return minimax optimal policy $\mathbf{f}$
    end
end
```

Many variants of this scheme exist. The pruning and addition of policies need not be done in real time, but can take place in some parallel background process. For instance, minimax regret w.r.t. $\mathcal{R}_t$ can be computed using a "lagging" set $\tilde{\Gamma}_{t\text{-}k}$ (the set may include some dominated policies and omit some nondominated policies) without detriment; in this case, the error would be determined by the error of the lagging approximate set. Update of $\tilde{\Gamma}$ can take place asynchronously: whenever a set of update operations has been completed relative to any $\mathcal{R}_{t\text{-}k}$, it can be used at stage t.

6.2 Nondominated Region Vertex Algorithm

The ability to add the most relevant new policies to the approximate set $\tilde{\Gamma}$ in our online procedure allows error in minimax regret to be reduced significantly while maintaining good online computational performance. We now describe a principled anytime algorithm for generat-

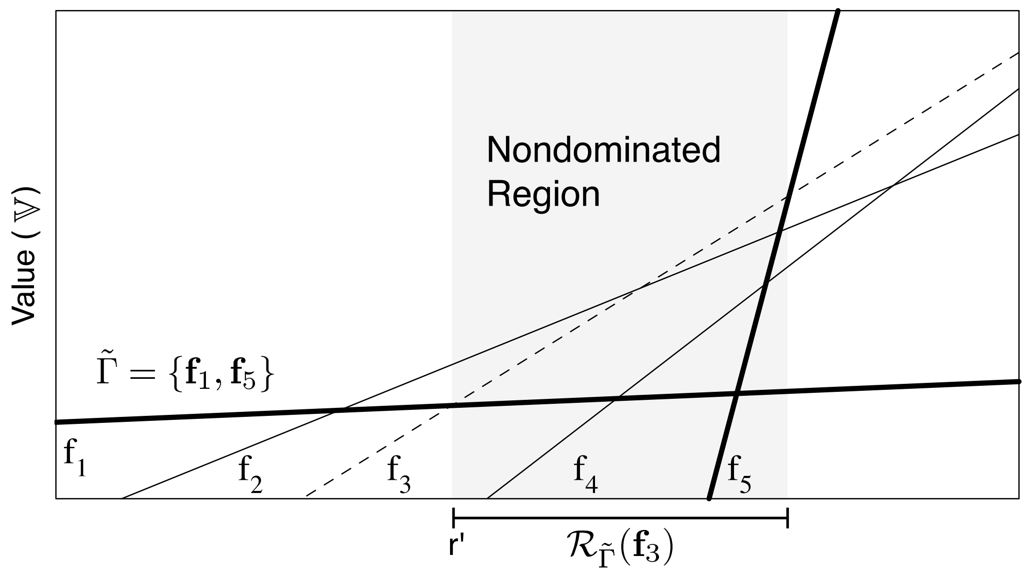

Nondominated Region
Value ($\mathbb{V}$)
$\tilde{\Gamma} = \{\mathbf{f}_1, \mathbf{f}_5\}$
f_1
f_2
f_3
f_4
f_5
r'
$\mathcal{R}_{\tilde{\Gamma}}(\mathbf{f}_3)$

Figure 6.2: Illustration of a nondominated region.

ing an approximate set $\tilde{\Gamma}$ which directly minimizes value function error. Our algorithm is an adaptation of Cheng's classic *linear support method* for POMDPs (1988). Rather than computing nondominated α-vectors over belief states, we compute policies that are nondominated w.r.t. uncertain reward. We note that unlike the heuristic technique for generating approximation sets $\tilde{\Gamma}$ proposed for πWitness in Section 3.7.3, our algorithm yields theoretical bounds on error, without which we could not offer the user a guarantee on maximum regret during elicitation.

Let $\mathcal{R}$ be some feasible set of reward functions (e.g., a reward polytope). Given a set $\tilde{\Gamma} \subseteq \Gamma(\mathcal{R})$, let $\mathcal{R}_{\tilde{\Gamma}}(\mathbf{f})$ be the *nondominated region* of policy $\mathbf{f}$ w.r.t. $\tilde{\Gamma}$:

$$\mathcal{R}_{\tilde{\Gamma}}(\mathbf{f}) \equiv \left\{ r \in \mathcal{R} \;\middle|\; \mathbf{f} \cdot \mathbf{r} \geq \mathbf{f}' \cdot \mathbf{r}, \ \forall\, \mathbf{f}' \in \tilde{\Gamma} \right\}, \tag{6.2}$$

that is the region of $\mathcal{R}$ for which $\mathbf{f}$ is the best policy in $\tilde{\Gamma}$. Figure 6.2 illustrates the nondominated region $\mathcal{R}_{\tilde{\Gamma}}(\mathbf{f}_3)$ of policy $\mathbf{f}_3$ with respect to the approximate set $\tilde{\Gamma} = \{\mathbf{f}_1, \mathbf{f}_5\}$ (depicted in bold). The nondominated region $\mathcal{R}_{\tilde{\Gamma}}(\mathbf{f})$ for any $\mathbf{f} \in \tilde{\Gamma}$ is a bounded, convex polytope—since we define the region by adding the linear constraints in Eq. (6.2) to an already bounded and convex polytope, $\mathcal{R}$.

In Section 3.7.3 we defined the value function induced by $\tilde{\Gamma}$ w.r.t. to the fixed reward $\mathbf{r}$ as:

$$\mathbb{V}_{\tilde{\Gamma}}(\mathbf{r}) = \max_{\mathbf{f}\in\tilde{\Gamma}} \ \mathbf{f}\cdot\mathbf{r}. \tag{3.27}$$

Note that $\mathbb{V}_{\tilde{\Gamma}}$ is piecewise linear and convex (PWLC) over $\mathbf{r} \in \mathcal{R}$. We define the *error* in $\mathbb{V}_{\tilde{\Gamma}}$ for fixed reward $\mathbf{r} \in \mathcal{R}$ as the difference between the approximate and exact value functions:

$$\epsilon_{\mathbb{V}}(\tilde{\Gamma}, \mathbf{r}) = \mathbb{V}_{\Gamma}(\mathbf{r}) - \mathbb{V}_{\tilde{\Gamma}}(\mathbf{r})$$

The error function $\varepsilon_{\mathbb{V}}(\tilde{\Gamma}, \mathbf{r})$ is convex over $\mathbf{r} \in \mathcal{R}_{\tilde{\Gamma}}(\mathbf{f})$, since error is defined as the difference between two parameterizations of $\mathbb{V}$, each of which is PWLC. Hence the maximum of $\varepsilon_{\mathbb{V}}(\tilde{\Gamma}, \mathbf{r})$ over the region $\mathcal{R}_{\tilde{\Gamma}}(\mathbf{f})$ must lie at a vertex of $\mathcal{R}_{\tilde{\Gamma}}(\mathbf{f})$. Formally, for some policy $\mathbf{f}$:

$$\epsilon_{\mathbb{V}}(\tilde{\Gamma}, \mathcal{R}_{\tilde{\Gamma}}(\mathbf{f})) = \max_{\mathbf{r}\in\mathcal{R}_{\tilde{\Gamma}}(\mathbf{f})} \mathbb{V}_{\Gamma}(\mathbf{r}) - \mathbb{V}_{\tilde{\Gamma}}(\mathbf{r}) \in \textit{VERTICES}\big(\mathcal{R}_{\tilde{\Gamma}}(\mathbf{f})\big)$$

For example, in Figure 6.2 the maximum over $\mathcal{R}_{\tilde{\Gamma}}(\mathbf{f}_3)$ is found at the vertex labelled $\mathbf{r}'$. Given any approximate set of nondominated policies $\tilde{\Gamma} \subseteq \Gamma$, the nondominated regions of all nondominated policies $\mathbf{f}' \in \tilde{\Gamma}$ cover the entire feasible reward set:

$$\mathcal{R} = \bigcup_{\mathbf{f}'\in\tilde{\Gamma}} \mathcal{R}_{\tilde{\Gamma}}(\mathbf{f}')$$

As a consequence, the maximum error $\varepsilon_{\mathbb{V}}(\tilde{\Gamma}, \mathbf{r})$ over $\mathcal{R}$ must lie at the vertex of the nondominated region of *some* $\mathbf{f} \in \tilde{\Gamma}$.

The *nondominated region vertex (NRV)* algorithm exploits this fact by computing error only at vertices of such regions, and adding (optimal) policies to $\tilde{\Gamma}$ only for those vertices with maximal error. The algorithm (detailed in Algorithm 6) begins with an initial nondominated policy $\mathbf{f}$ (optimal for some arbitrary $\mathbf{r} \in \mathcal{R}$) in $\tilde{\Gamma}$. It adds a policy by: (a) computing $E_{\tilde{\Gamma}}$, the

Algorithm 6: Nondominated Region Vertex algorithm

```
   Let $\delta$ be allowable error, and $\mathbf{r}_0$ some vertex of $\mathcal{R}$ ;
   $\tilde{\Gamma} \leftarrow \emptyset$   *subset of nondominated policies* ;
   $E \leftarrow \{\mathbf{r}_0\}$   *vertices of the nondominated regions* ;
   $\varepsilon_{\mathbb{V}}(\mathbf{r}_0) \leftarrow \infty$   *(arbitrary) initial error* ;
   $E' \leftarrow \emptyset$   *vertices with error below threshold* $\delta$ ;
   **while** $E - E' \neq \emptyset$ **do**
1      $\mathbf{r}' \leftarrow \mathrm{argmax}_{\mathbf{r}\in E-E'}\, \varepsilon_{\mathbb{V}}(\mathbf{r})$ ;
2      $\mathbf{f}_{\mathbf{r}'} \leftarrow \mathrm{argmax}_{\mathbf{f}\in\mathcal{F}}\, \mathbf{f}\cdot\mathbf{r}'$ ;
       $\tilde{\Gamma} \leftarrow \tilde{\Gamma} \cup \{\mathbf{f}_{r'}\}$ ;
       $E \leftarrow E \cup Vertices(\mathcal{R}_{\tilde{\Gamma}}(\mathbf{f}_{r'}))$ ;
       $E' \leftarrow E' \cup \{\mathbf{r}'\}$ ;
       **foreach** $\mathbf{r} \in E - E'$ **do**
3          $\varepsilon_{\mathbb{V}}(\mathbf{r}) \leftarrow \mathbb{V}(\mathbf{r}) - \mathbb{V}_{\tilde{\Gamma}}(\mathbf{r})$ ;
           **if** $\varepsilon_{\mathbb{V}}(\mathbf{r}) \leq \delta$ **then**
               $E' \leftarrow E' \cup \{\mathbf{r}\}$ ;
           **end**
       **end**
   **end**
```

set of vertices of the nondominated regions of $\tilde{\Gamma}$; (b) computing the optimal policy $\mathbf{f}_\mathbf{r}$ for each $\mathbf{r} \in E_{\tilde{\Gamma}}$; and (c) selecting the policy that offers the greatest improvement, i.e., such that the error $\mathbf{f}_\mathbf{r}\cdot\mathbf{r} - \max_{\mathbf{g}\in\tilde{\Gamma}} \mathbf{g}\cdot\mathbf{r}$ is maximal. The selected policy is added to $\tilde{\Gamma}$ and the process repeated until the maximum error at any vertex falls below an acceptable threshold.

As in with the algorithm developed by Cheng (1988), many efficiencies are exploited that enhance the high-level description in Algorithm 6. For example, caching is used to eliminate duplication in computing max error (see line 1), finding the optimal policy (line 2) and computing error at each new vertex (line 3). We use the LRS backward search algorithm for vertex enumeration (Avis, 2000)—the algorithm operates in a manner similar to the simplex method (Chvátal, 1983) using a *pivot rule* to move between vertices; however, rather than moving towards an optimal vertex, the enumeration algorithm chooses pivots that guarantee all vertices are visited.

Secondary information generated by NRV can also be leveraged to speed up the online adjustment of $\tilde{\Gamma}$. By storing the vertex $\mathbf{r}$ at which each policy $\mathbf{f} \in \tilde{\Gamma}$ was found to be optimal, we can quickly determine whether $\mathbf{f}$ remains nondominated by testing whether $\mathbf{r}$ remains feasible. If $\mathbf{r} \in \mathcal{R}'$ (i.e., satisfies the new constraints that refine $\mathcal{R}$), then $\mathbf{f}$ remains nondominated. If $\mathbf{r} \notin \mathcal{R}'$, then we resort to LP (6.1).

6.2.1 Empirical Evaluation

We test the NRV algorithm on small MDPs and compare it to the πWitness algorithm (developed in Section 3.7). We test both methods on small IRMDPs with *factored, additive reward functions*. We assume each reward function is naturally calibrated obviating the complications that arise from representing sub-reward functions as products of local value functions and scaling constants. A state $\mathbf{x} = \langle x_1, x_2, \ldots, x_7 \rangle$ is composed of 7 binary variables, yielding $|\mathbf{X}| = 128$. We use two different reward functions: the first $r(\mathbf{x}) = r_1(x_1) + r_2(x_2) + r_3(x_3)$ with dimension 6; and the second $r(\mathbf{x}) = r_1(x_1) + r_2(x_2) + r_3(x_3) + r_4(x_4)$ with dimension 8. For each MDP we generate the transition model and uncertain reward function $\mathcal{R}$ using the procedure from Section 3.2. We generate 20 MDPs for each size of reward dimension, and run both NRV and πWitness to completion, generating *all* nondominated policies.

Figures 6.3 and 6.4 show the average runtime of each algorithm. NRV is more efficient than πWitness on small MDPs—it takes NRV an order of magnitude longer on average to generate 2000 policies (221 minutes for NRV vs. 22 minutes for πWitness). The performance gap narrows as reward dimensionality increases—NRV requires roughly double the amount of time to generate 4900 policies (1194 minutes for NRV vs. 618 minutes for πWitness). The primary advantage of NRV is the availability of an error bound $\varepsilon_{\mathbb{V}}(\tilde{\Gamma}, \mathcal{R})$ at each iteration.

Figure 6.5 shows that the error $\varepsilon_{\mathbb{V}}$ drops quickly with each added nondominated policy (note the log scale). For example, $\varepsilon_{\mathbb{V}}(\tilde{\Gamma}, \mathcal{R})$ is reduced to well under 1.0% of its initial value after only 500 policies, and to nearly 0.1% after 2000 policies. A small set $\tilde{\Gamma}$ of nondominated policies can be used to quickly approximate minimax regret as discussed above. We need only compute $\tilde{\Gamma}$ once (offline), prior to elicitation. Minimax regret, conversely, needs to be computed *repeatedly* and *online*, since it is integral to our elicitation scheme. Thus offline

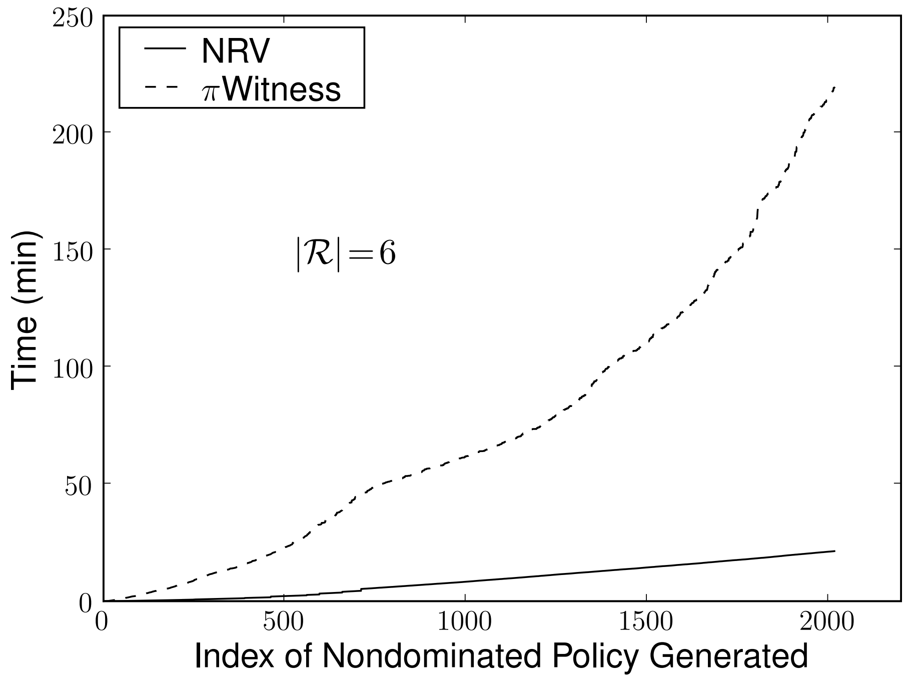


Figure 6.3: Time to generate nondominated policies (20 instances) for both NRV and πWitness: $|\mathcal{R}|$=6.

computation of a small $\tilde{\Gamma}$ with small error can greatly enhance online performance. For this reason, the extensive offline computation required for computing nondominated sets is not necessarily problematic. However, we can further leverage NRV when we allow minimax regret approximation in the online setting.

We now examine how NRV, used in conjunction with constraint generation for fast, approximate solution of minimax regret, works in the context of our online optimization scheme. Recall our online model provides the ability to improve the quality of an approximate solution during elicitation, while maintaining tractability. We demonstrate this potential by revisiting the COACH system for guiding persons with dementia through daily living tasks (Boger et al., 2006).

We review the main elements of the model—described in detail in Section 5.5.1—before setting out some modifications for our current evaluation. The system guides the user through a task with ℓ steps; and can issue one of k prompts at increasing levels of intrusiveness, or can call a caregiver to assist the person in task completion. The state is defined by three variables $\mathcal{S} = \langle T, D, F \rangle$ where $T \equiv \{0, 1, \ldots, \ell\}$ is the number of tasks steps successfully

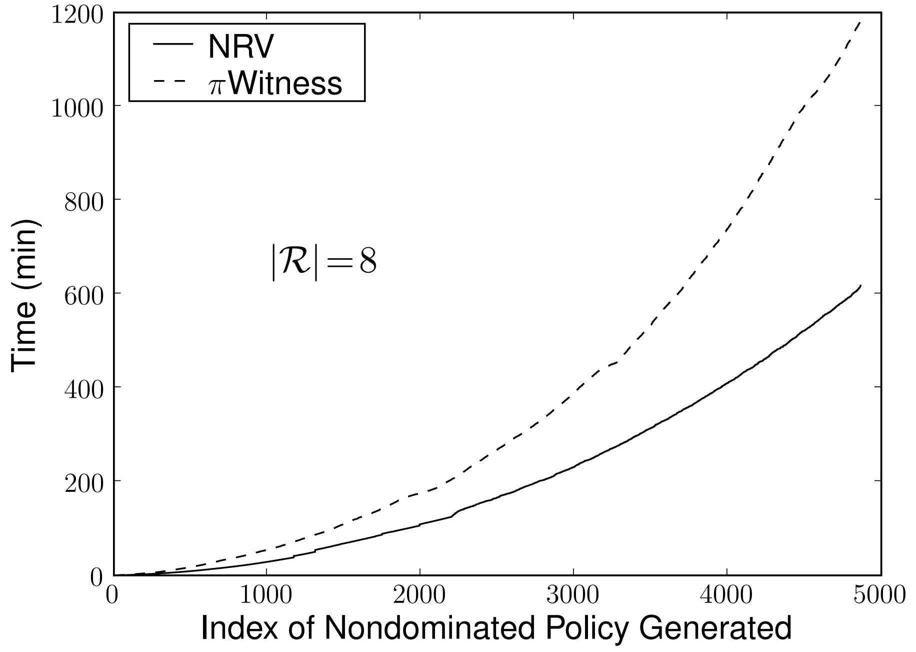


Figure 6.4: Time to generate nondominated policies (20 instances) for both NRV and πWitness: $|\mathcal{R}|$=8.

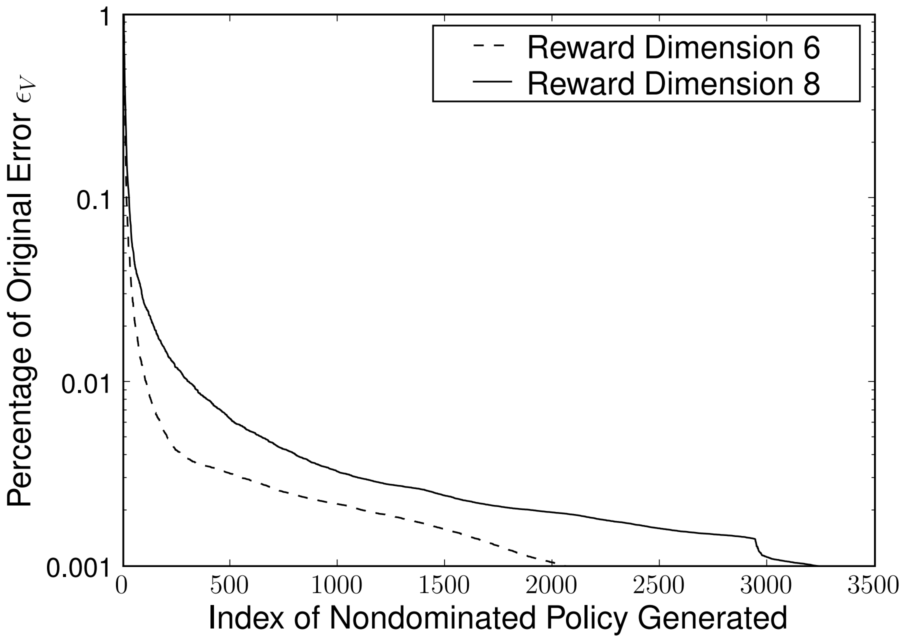


Figure 6.5: Error $\varepsilon_{\mathbb{V}}$ as function of NRV policies generated (20 random IRMDPs, log scale).

completed, $D \equiv \{0, 1, 2, 3, 4, 5+\}$ is the *delay* (time taken during the the current step); and $F \equiv \{0, 1, \ldots, k\text{-}1\}$ tracks whether a prompt at a specific level was attempted at the current step and failed to immediately get the person to the next step. The reward function is as follows:

$$\begin{aligned} r(t, d, f, a) &= r_g(t) + r_d(d) + r_p(a) \\ &= \lambda_g \upsilon_g(t) + \lambda_d \upsilon_d(d) + \lambda_p \upsilon_p(a). \end{aligned}$$

Where $r_g(t)$ is a large positive reward when $t = \ell$ for completing the task and is zero when $t < \ell$; $r_d(d = 0)$ is a (small) positive reward for progressing to step t (indicated by d being reset to zero); $r_d(d > 0)$ is a small negative reward for delay in completing a step; and $r_p(a)$ is the negative cost associated with prompting the person. The precise values of the rewards are not known but must be elicited from a caregiver.

The first key modification is as follows: we assume the scaling constants $\boldsymbol{\lambda}$ that calibrate the local value functions are already known. While this is not necessarily a realistic assumption, it allows us to focus our evaluation on our current approach to online minimax regret computation. Removing uncertainty over $\boldsymbol{\lambda}$ puts aside the computational complications that result from the interaction between $\boldsymbol{\upsilon}$ and $\boldsymbol{\lambda}$ that were observed in Section 5.3, allowing for us to employ the computational machinery (i.e., the NRV algorithm) developed in this chapter without modification. The task of developing approaches for leveraging policies that are non-dominated w.r.t. to a quadratic parameterization of reward (i.e., $\mathbf{r} = \boldsymbol{\lambda\upsilon}$) is important future work.

The second, more minor, modification is to scale up the parameter settings slightly so as to generated larger IRMDPs; we increase the number of steps ℓ and prompt levels k. We set $\ell = 14$, $k = 6$ and create an IRMDP by setting initial reward bounds to construct $\mathcal{R}$ in a manner similar to the random IRMDPs discussed in the previous section. The resulting IRMDP has size $|\mathbf{X}||\mathcal{A}| = 3012$ and reward dimensionality $|\mathcal{R}|$=12. For comparison, the COACH example from Section 5.5.1 has parameter settings $\ell = 14$, $k = 6$, resulting in $|\mathbf{X}||\mathbf{A}| = 960$.

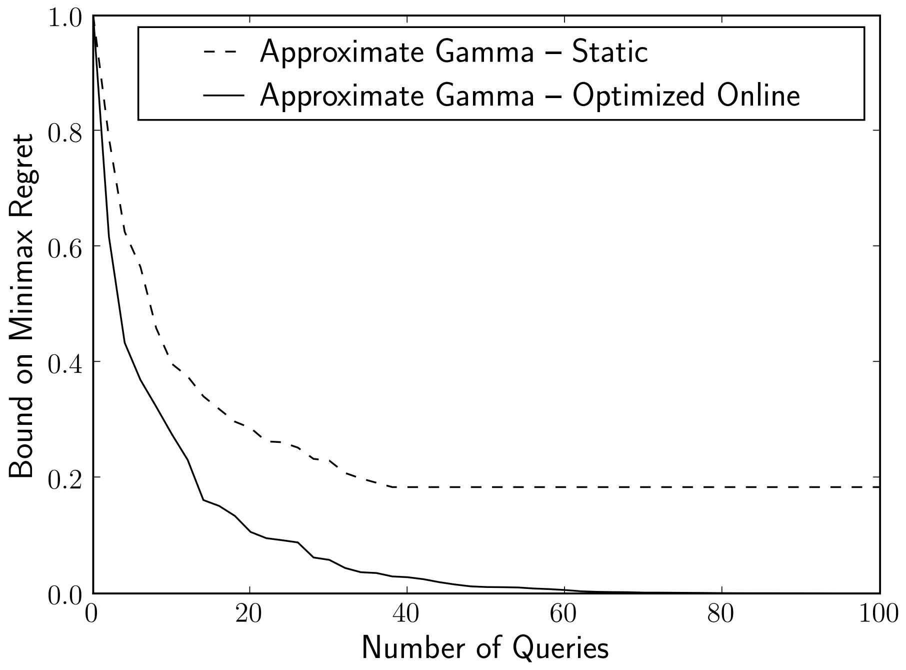


Figure 6.6: Upper Bound on Minimax Regret (as proportion of initial regret) during Elicitation for the COACH model.

Reward elicitation is performed with bound queries selected using the current solution heuristic. Full details of this approach can be found in Sections 5.2 and 5.4. Generally, at each step t of elicitation: 1) minimax regret is computed w.r.t. the current reward polytope $\mathcal{R}_t$; 2) the minimax regret solution is used to select a query using the CS heuristic; 3) the query response is used to refine $\mathcal{R}_t$ to produce $\mathcal{R}_{t+1}$. Elicitation terminates once max regret τ reaches an acceptable level. When a nondominated set $\tilde{\Gamma}$ is used to approximate the minimax optimal policy, we introduce a termination condition that takes into account this approximation and NRV algorithm's accompanying error bound: $MMR(\tilde{\Gamma}, \mathcal{R}_t) + \varepsilon_{\mathbb{V}}(\tilde{\Gamma}, \mathcal{R}_t) \leq \tau$.

During the offline phase, we use the NRV algorithm generate the initial $\tilde{\Gamma}$ with the following criterion in mind. We wish to allow for interactive response times during elicitation, so we choose the size of $\tilde{\Gamma}$ so that $MMR(\tilde{\Gamma}, \mathcal{R})$ takes no more than one second to compute. This results in $\tilde{\Gamma}$ containing less than 5% of all nondominated policies. We further assume that during elicitation there are ten seconds available while waiting for a user response to perform online optimization (pruning and addition) of $\tilde{\Gamma}$.

During elicitation we compute minimax regret using: a static set $\tilde{\Gamma}$ with error $\varepsilon_{\mathbb{V}}$; and a dynamic set $\tilde{\Gamma}'$ with decreasing error $\varepsilon'_{\mathbb{V}}$ that is optimized online by pruning and adding policies during the ten-second period provided by response latency. Figure 6.6 shows the upper bounds $MMR(\tilde{\Gamma}, \mathcal{R}) + \varepsilon_{\mathbb{V}}(\tilde{\Gamma}, \mathcal{R})$ and $MMR(\tilde{\Gamma}', \mathcal{R}) + \varepsilon_{\mathbb{V}}(\tilde{\Gamma}', \mathcal{R})$ on max regret produced by the static set and online-optimized set, respectively; it is shown as the percentage of the initial upper bound on minimax regret prior to the start of elicitation. We see that online optimization of the nondominated set provides a tremendous benefit in terms of elicitation. First, without online adjustment of $\tilde{\Gamma}$, it is impossible to find the optimal policy: indeed, the static set stalls after roughly 40 queries with minimax regret that is still roughly 18% of the initial regret level. Online optimization of $\tilde{\Gamma}$ allows discovery of the optimal policy with approximately 60 queries (this is about 5 simple bound queries per reward parameter). Just as importantly, if an approximately optimal policy is desired, the online-optimized $\tilde{\Gamma}$ reduces minimax regret to 20% of its initial levels with only about 12 queries on average, while the static approach requires almost 35 queries. In this example, a small $\tilde{\Gamma}$ with less than 5% of all nondominated policies enables effective reward elicitation, quickly reducing the approximation error to zero if $\tilde{\Gamma}$ is optimized online. This demonstrates the power of our online approach. A very small set of nondominated policies is needed for fast online computation; but a static set of the required size does not admit an approximation of suitable quality. Pruning newly dominated policies during elicitation and adding new policies using NRV allows one to maintain online feasibility while reducing provable max regret to zero, while supporting effective elicitation (both in terms of number of queries and interactive response time).

In the next section, we continue to investigate the performance of the NRV algorithm by comparing to it to some related work that exploits the geometric properties of the reward polytope.

6.3 A Comparison to the Geometric Traversal Algorithm

In their recent paper Oh and Kim (2011a) develop both an exact and approximate algorithm for generating the set of nondominated policies. We briefly discuss the exact algorithm—which offers a legitimate improvement over the πWitness algorithm—before detailing an empirical evaluation of the approximate GT algorithm that undermines its value for the focus of this chapter: online minimax regret computation.

6.3.1 Exact GT

The *geometric traversal* (GT) algorithm finds the exact set of nondominated policies by exploiting the inverse reinforcement learning (IRL) constraints observed by Ng (2000) that characterize the region of reward space where a given policy is optimal. These constraints partition reward space into the regions defined by policy optimality, where each region corresponds to a nondominated policy. The geometric traversal algorithm operates by treating each region as a graph node and performing an exhaustive graph traversal. The approach yields computational complexity linear in the number of nondominated policies (Oh and Kim, 2011a); this improves upon the theoretical complexity of the πWitness algorithm (developed in Section 3.7 which is quadratic in the number of nondominated policies.

6.3.2 Approximate GT

The approximate variant of geometric traversal algorithm is more relevant to the discussion in this chapter. The essential idea is as follows: at each iteration, the algorithm traverses a subset of adjacent reward regions that are encountered while moving along a randomly sampled straight line. Figure 6.7 details the algorithm as described by Oh and Kim (2011a). Like the exact GT algorithm, the approximate GT algorithm yields a runtime that is linear in the number of nondominated policies found (unlike both πWitness and NRV which are super-linear in the number of nondominated policies). However, the runtime of the approximate GT algorithm constitutes the *offline* step of minimax regret computation for which efficiency is less crucial. More important is the size of approximate set of nondominated policies and the quality of the approximation. In this regard, results from Oh and Kim (2011a) indicate that πWitness outperforms the approximate GT algorithm (we later show that NRV outperforms both πWitness and approximate GT).

Algorithm A: Geometric Traversal Algorithm

```
  begin
      $\mathbf{r} \leftarrow$ some arbitrary $\mathbf{r} \in R$
      $\mathbf{f} \leftarrow \mathbf{findOptPolicy}(\mathbf{r})$
      $\Gamma \leftarrow \{\mathbf{f}\}$
      agenda $\leftarrow \{\langle \mathbf{r}, \mathbf{f} \rangle\}$
      while agenda is not empty do
          $\langle \mathbf{r}, \mathbf{f} \rangle \leftarrow$ next item in agenda
*         $H \leftarrow \mathbf{findRewardOptRgn}(\mathbf{r}, \mathbf{f})$
          for $h \in H$ do
              $\mathbf{r}' \leftarrow \mathbf{findAdjRewardFn}(h, H)$
              if $\mathbf{r}'$ is found then
                  $\mathbf{f}' \leftarrow \mathbf{findOptPolicy}(\mathbf{r}')$
                  if $\mathbf{f}' \notin \Gamma$ then
                      add $\mathbf{f}'$ to $\Gamma$
                      add $\langle \mathbf{r}', \mathbf{f}' \rangle$ to agenda
```

Algorithm B: Approximate Geometric Traversal Algorithm

```
  begin
      $\Gamma \leftarrow \{\}$
      agenda $\leftarrow \{\}$
      while $\Gamma$ is not sufficiently gathered do
          $\mathbf{r} \leftarrow$ some arbitrary $\mathbf{r} \in R$
          $l \leftarrow$ arbitrary straight line passing through $\mathbf{r}$
          $\mathbf{f} \leftarrow \mathbf{findOptPolicy}(\mathbf{r})$
          add $\mathbf{f}$ to $\Gamma$
          add $\langle \mathbf{r}, \mathbf{f} \rangle$ to agenda
          while agenda is not empty do
              $\langle \mathbf{r}, \mathbf{f} \rangle \leftarrow$ next item in agenda
*             $H \leftarrow \mathbf{findRewardOptRgn}(\mathbf{r}, \mathbf{f})$
              $\{\mathbf{r}_1, \mathbf{r}_2\} \leftarrow$ find two intersections from $H$
              for $\mathbf{r}' \in \{\mathbf{r}_1, \mathbf{r}_2\}$ do
                  $\mathbf{f}' \leftarrow \mathbf{findOptPolicy}(\mathbf{r}')$
                  add $\langle \mathbf{r}', \mathbf{f}' \rangle$ to agenda
                  if $\mathbf{f}' \notin \Gamma$ then
                      add $\mathbf{f}'$ to $\Gamma$
```

Figure 6.7: Pseudocode for the Geometric Traversal Algorithm and the Approximate Geometric Traversal Algorithm. The line marked (*) in each algorithm computes the IRL constraints, represented as the polytope H.

Specifically, the authors compare approximate GT to πWitness on randomly generated MDPs and measure the number of nondominated policies required to yield an empirical minimax regret error of 10%, 5% and 1%. On the largest MDP examined, the approximate GT algorithm required an average of 729.6 policies to reach 5% error where πWitness only required an average of 72.2 (Oh and Kim, 2011a). We perform our own investigation to corroborate these findings and include a comparison with the performance of the NRV algorithm. As a baseline we also include a *pure sampling* algorithm that operates as follows: at each iteration we sample a random reward point from the feasible reward polytope, compute the optimal policy for that reward and, if the optimal policy is not already in the nondominated set, we add the new policy.

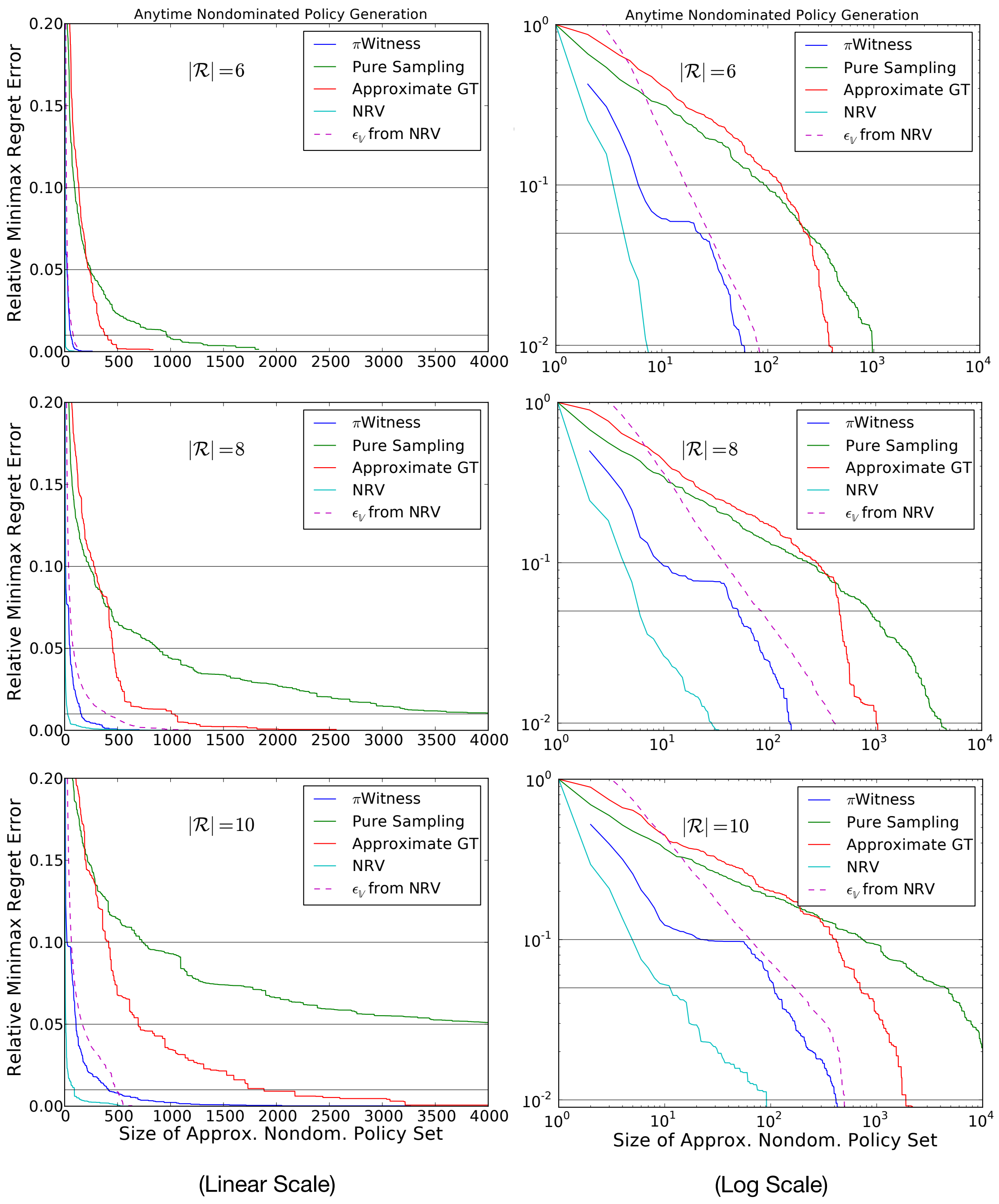


Figure 6.8: Reduction in (empirical) relative minimax regret error as policies are generated; linear scale in left column, log scale in right column.

3.3 Experiments

Our experimental procedure randomly generates 50 MDPs with sparse transition functions using the procedure described in Section 3.2. Each MDP has 5 actions and states are composed of 6 binary variables (yielding 128 states). We varied the dimension of the imprecise reward function generating MDPs with reward dimension $|\mathcal{R}| = 6$, 8, and 10. For each MDP we run the four nondominated policy generation algorithms: Pure Sampling, Approximate GT, πWitness, and NRV. At each iteration we use the partial set of nondominated policies generated by each algorithm to approximate minimax regret and we measure the relative error of the approximation (w.r.t. exact minimax regret computed using the MIP from Section 3.1).

Figure 6.8 shows results (averaged over 50 runs) comparing the reduction in relative MMR error (i.e., MMR error / exact MMR) of each algorithm as nondominated policies are generated (the right column of the figure shows the same results with a log scale). Error levels of 10%, 5% and 1% percent are marked with horizontal lines. At each iteration the NRV algorithm provides a bound on the value error, $\epsilon_{\mathbb{V}}$ induced by the approximate set of nondominated policies at each iteration, which in turn bounds the error of approximate MMR. We also plot the value error as a proportion of MMR (i.e. $\epsilon_{\mathbb{V}}$ / exact MMR) using a dashed line.

The experimental results show that the NRV algorithm makes the best use of the nondominated policies generated, reducing relative MMR error using significantly fewer policies than all other algorithms. While not as efficient as NRV, the πWitness algorithm in turn significantly outperforms the approximate GT algorithm. Our results reflect the experiments of Oh and Kim (2011a): on average the πWitness algorithm reduces relative MMR error to 5% using an order of magnitude fewer policies (NRV uses roughly two orders of magnitude fewer policies than approximate GT to perform the same feat). Interestingly, the performance of the approximate GT algorithm is initially worse than the pure sampling algorithm, and it is only after some number of nondominated policies is generated that the line sampling approach of the approximate GT algorithm is able to match and outperform simple pure sampling.

It is worth emphasizing that the relative MMR error being measured must be computed ex-post; only the NRV algorithm can offer a guarantee on MMR error *during* execution (using the derived error $\epsilon_{\mathbb{V}}$). Figure 6.8 shows that the bound on MMR error computed by NRV (marked by the dashed line in the figures) is reduced more efficiently than the empirical ex-post error of the approximate GT algorithm.

6.4 Summary and Conclusions

We have presented a method for computing approximate, robust solutions to *imprecise-reward* MDPs (IRMDPs) in the context of online reward elicitation. The NRV algorithm generates approximate sets of nondominated policies with provable error bounds, which can be leveraged to efficiently approximate minimax regret using the constraint generation method developed in Section 3.1.

We have shown how online optimization of the nondominated set, as reward knowledge is refined, allows regret to quickly decrease to zero with only a small fraction of all nondominated policies. An empirical analysis on the COACH domain demonstrates the value of our online approach. Taken together the results of this chapter remove a significant computational barrier to online reward elicitation for MDPs.

For IRMDPs with more than a handful of states and actions, we recommend using the NRV algorithm as a first step to gauge whether there is sufficient structure to support minimax regret computation using nondominated policies. As new nondominated policies are generated by NRV and added to the set $\tilde{\Gamma}$, the runtime of the minimax regret approximation using the $\tilde{\Gamma}$ increases while the approximation error goes down. A practitioner can select the desired balance between speed and accuracy when both measures are within acceptable ranges. In some cases, such as the website optimization domain explored in Chapter 7, the structure of IRMDP admits small and complete sets of nondominated policies that may be used to compute minimax regret exactly. For larger IRMDPs without sufficient structure, the offline generation of nondominated policies using NRV may prove computationally prohibitive. In such cases, the approximation methods that developed in Section 3.5 can be used to generated both lower and upper bounds on minimax regret.

A connection exists between our approach to identifying nondominated policies and work on solving a restricted class of decentralized MDPs (DEC-MDPs) that feature transition and observation independence (Becker, Zilberstein, Lesser, and Goldman, 2004). In this setting agents can be viewed as having nearly independent "local MDPs" that are coupled only by a joint reward function. The goal in this work is to cooperatively maximize cumulative reward.

Several recent approaches for solving such DEC-MDPs involve identifying the set of policies for each agent that are nondominated with respect to the space of potential policies chosen by the other agents (Becker et al., 2004; Petrik and Zilberstein, 2009). With some adaptation, our NRV algorithm could be used to solve transition independent DEC-MDPs. Likewise the *successive approximation* algorithm for DEC-MDPs (Petrik and Zilberstein, 2009)—which iteratively finds policies that are nondominated with respect to another agent's policies and admits an anytime error bound—could potentially be adapted to generating policies that are nondominated with respect to reward.

The NRV algorithm was inspired by a related approach to solving POMDPs (Cheng, 1988), as was the πWitness algorithm (Kaelbling et al., 1998). Future work that continues to investigate relevant POMDP algorithms that intelligently generate and managing the set of α-vectors (the POMDP analog of nondominated policies) could yield further algorithmic improvements. For instance, point-based value iteration (PBVI) algorithms (Pineau, Gordon, and Thrun, 2003; Spaan and Vlassis, 2005; Pineau, Gordon, and Thrun, 2006) maintain a subset of α-vectors that are nondominated w.r.t. small subset of belief points. When this set is well chosen, the resulting α-vectors constitute a good approximation. An analog in our domain would limit nondominated policy generation to a subset of well chosen reward points; however, working out the details of what constitutes a "well chosen" set of reward points in our domain remains a challenge, especially when we consider that—unlike belief space in POMDPs—the set of feasible reward points is not static, and is refined as information is gained during elicitation.

6.4.1 Contributions

- The nondominated-region vertex (NRV) algorithm for generating nondominated policies which provides an anytime bound on approximation error for minimax regret (Regan and Boutilier, 2011b)
- A method for adjusting the set of nondominated policies online, speeding up computation and improving the quality of approximation (Regan and Boutilier, 2011b)

Chapter 7

Applications

7.1 Background

Data driven web companies such as Google and Amazon.com are becoming increasingly sophisticated in their approach to improving their web-based products. Webpages are constantly redesigned in order to optimize various objectives such as user engagement or item sales. The approach used by most modern companies for evaluating the impact of small changes to webpages is referred to A/B, or *version-testing*. The simplest variant compares two versions of webpage[1] by randomly assigning website visitors to a webpage version while recording the differences in user behaviour.

[1]In some cases companies simultaneously compare *many* more than two examples: Google famously compared 41 shades of blue (Holson, 2009) in order to select the colour of a toolbar.

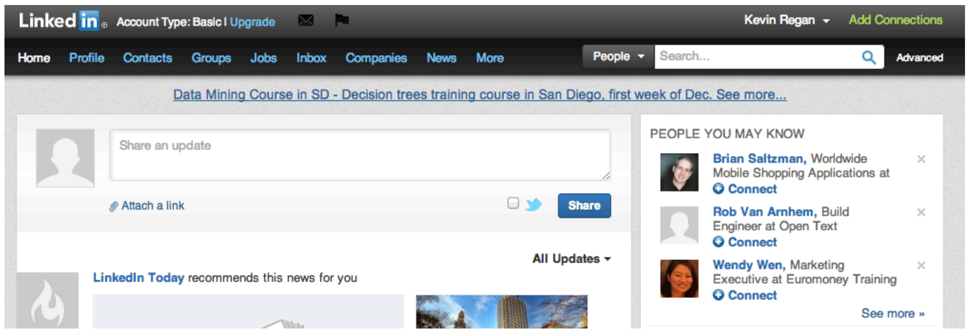
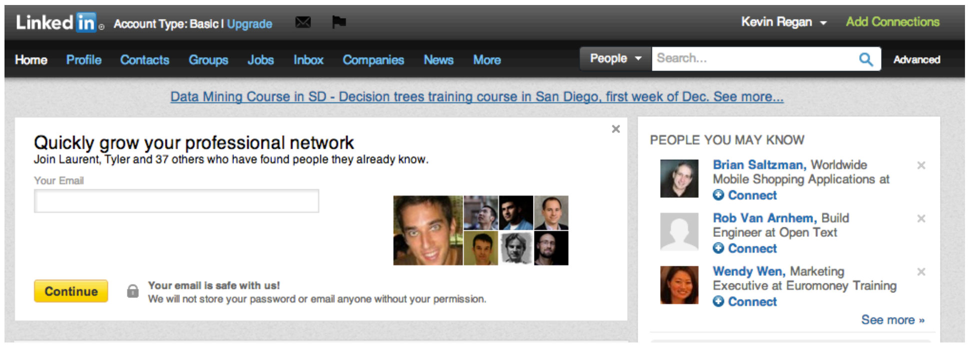

Linked in
Account Type: Basic | Upgrade
Kevin Regan
Add Connections
Home
Profile
Contacts
Groups
Jobs
Inbox
Companies
News
More
People
Search...
Advanced
Data Mining Course in SD - Decision trees training course in San Diego, first week of Dec. See more...
Share an update
Attach a link
Share
All Updates
LinkedIn Today recommends this news for you
PEOPLE YOU MAY KNOW
Brian Saltzman, Worldwide Mobile Shopping Applications at
Connect
Rob Van Arnhem, Build Engineer at Open Text
Connect
Wendy Wen, Marketing Executive at Euromoney Training
Connect
See more »

Figure 7.1: Version A of a User's LinkedIn homepage.



Data Mining Course in SD - Decision trees training course in San Diego, first week of Dec. See more...

Quickly grow your professional network

Join Laurent, Tyler and 37 others who have found people they already know.

Your Email

Continue

Your email is safe with us!

We will not store your password or email anyone without your permission.

PEOPLE YOU MAY KNOW

Brian Saltzman, Worldwide Mobile Shopping Applications at
Connect

Rob Van Arnhem, Build Engineer at Open Text
Connect

Wendy Wen, Marketing Executive at Euromoney Training
Connect

See more »

Figure 7.2: Version B of a User's LinkedIn homepage.

Figures 7.1 and 7.2 visualize one example of version-testing performed on a user's homepage at LinkedIn.com. One of LinkedIn's primary goals is to grow its social network by encouraging users to add their professional contacts. The LinkedIn homepage typically displays a right sidebar titled "people you may know" that lists potential contacts. Version A (shown in Figure 7.1) includes this sidebar along with unrelated functionality in the main pane allowing the user to post updates. Version B (shown in Figure 7.2) replaces the contents of the main pane with a personalized message exhorting the user to "quickly grow your professional network".

A typical version test would randomly assign users among each version and measure the resulting user behaviour. This measurement typically focuses on the impact each version has on a goal such as increasing the number of professional contacts added by the user. Formally, let $\mathcal{W}$ be the set of versions and let $\mathcal{O}$ be the set of outcomes when a user is assigned to a particular version $w \in \mathcal{W}$. The set of outcomes often corresponds to the set of hyperlinks that may be clicked by the user in the webpage version. A completed version test induces a function vt mapping each version to a distribution over outcomes $vt : \mathcal{W} \rightarrow \Pi(\mathcal{O})$.

Returning to our example, we might expect that the additional message displayed by Version B will increase the likelihood that users will take some action to add contacts. However, we should also expect Version B to decrease the likelihood that a user will "share an update" (since this functionality is not present in Version B). Given a user's fixed attention span, directing a user to undertake either behaviour (i.e., sharing an update or adding a contact) may also decrease the likelihood that they will remain on the site and engage in other desired behaviour (e.g.,upgrading to a paid account). Choosing a version often involves balancing trade-offs between multiple goals. A decision theoretic approach to these trade-offs can be adopted by quantifying the strength preference for each goal and choosing the version that offers highest expected value.

Current industry practice performs a version-testing on a single page in isolation (though many version tests may be simultaneously operating) with purpose of optimizing one or more objectives. However, the results of multiple version tests can be stitched together to create a stochastic model of user behaviour as they navigate a *sequence* of webpages. Given this model and a reward function that precisely quantifies preferences over website goals, the problem of choosing versions so as to maximize expected reward is naturally modeled as a Markov decision process.

In this domain specifying the reward function involves precisely quantifying the relative value of multiple website goals. To alleviate this burden we apply our reward elicitation framework and demonstrate that we can efficiently find the optimal version of each page while specifying a relatively small amount of high-impact information about the reward function.

The rest of this chapter outlines the MDP model and the assessment of the effectiveness of reward elicitation in this domain. Empirical analysis using a number of public datasets show that given loose bounds on reward, the initial policy computed can improve website value by at least 2–24 percent, and that after a brief elicitation the value of the website is improved by 15–38 percent (details can be found in Section 7.5).

7.2 Model

We focus on version-testing that ignores any user specific information and uniformly assigns users to versions. The result of version-testing yields a model measuring the impact of a version in aggregate across all users. Thus, the Markov decision process that we describe captures expected behaviour of this aggregate user as they navigate the website. More advanced version-testing that assigns versions based on user demographics can be captured by the proceeding MDP model in a straightforward manner at the expense of additional state space and transition function complexity. We discuss some related work in this regard in Section 7.6. Another direction in which the model can be extended is the addition of "user state" variables capturing relevant aspects of the users interaction with the website. For example, a *bought-premium-membership* variable could indicate that a user had already purchased a premium membership on the website; the reward function could then be tailored to only give reward for a user signing up for a premium membership when *bought-premium-membership* is *false*. Extending the model in this manner is also straightforward; however, we omit "user state" modeling from our experiments due to the lack of data for constructing reasonable and realistic models of user state (and accompanying dynamics).

We define the states S of the MDP to correspond directly to the pages of the website to be optimized. Given the current page s requested by a user, the dynamics of the MDP predict the next page s' that the user will visit.[1] Let $S_{vt} \subset S$ be the set of pages on which version-testing is performed. For each page $s \in S_{vt}$, let $A(s)$ be the set of versions that have been tested; the choice of a version $a \in A(s)$ constitutes an action in our model. For $s \in S \setminus S_{vt}$, there is a single default action $A(s) = \{a_0\}$ that represents choosing the single static version of the webpage. Each version test measures user behaviour in terms of clicks on hyperlinks, resulting in navigation to the subsequent page $s' \in S$. Thus the results of the version-testing are captured by $P_{vt} : S \times A \to \Pi(S)$; where $P_{vt}(s'|s, a)$ is the probability that a user will navigate to page s' having been presented with version a of page s.

Independent of any version-testing, most websites log user activity that may be used to construct a stochastic model of user behaviour w.r.t. to the underlying *static* website. Let $P_{static} : S \setminus S_{vt} \to \Pi(S)$ capture this model; where $P(s'|s)$ is the probability that the user will navigate from (static) page s to page s'. We define the full transition function P of the MDP as

[1]Not all user behaviour takes the form of explicitly navigating to another webpage; however, all user interaction with website involves an HTTP request, and each request can be mapped to an implicit webpage so as to included in our model.

follows:

$$P(s'|s,a) = \begin{cases} P_{vt}(s'|s,a) & \text{if } s \in S_{vt} \\ P_{static}(s'|s) & \text{otherwise} \end{cases}$$

Given a user request s and the response that chooses a version a to send to the user, the transition to the next state is understood to be triggered by the user clicking a link somewhere on the current version a of the webpage, which initiates the next webpage request. Beyond clicking on hyperlinks, most user interactions with a webpage can be captured as HTTP requests and incorporated as subsequent states in our model.

Dynamics

Different versions of a webpage vary its HTML and hyperlink structure affecting the probability that a user will click on a link, requesting the next page s'. We assume that version-testing has already been performed and that these probabilities have been measured and recorded. Let $\Pr(s' \mid s, a)$ be the transition function specifying the probability of a user requesting webpage s' given that they had been served version a of webpage s.

To model the user entering and exiting the site, we designate two implicit pages s_{enter} and s_{exit}. We set the starting state distribution $\beta(s_{enter}) = 1$ and and define $P_{static}(s'|s_{enter})$ to be the probability that a user will begin their session on page s'. For $s \in S_{vt}$, $P_{vt}(s_{exit}|s,a)$ is the probability that the user does not click on any links in the version a of page s.; similarly $P_{static}(s_{exit}|s)$ is the probability that the user clicks on no further links in the static page $s \in S \setminus S_{vt}$.

Reward

We focus on websites that have multiple goals for the user interaction. The reward function $r : S \to \mathbb{R}$ quantifies the value a user visiting some page $s \in S$. We assume that website optimization is focused on funnelling users to a small set of high-value webpages. We do not attempt to elicit reward for pages outside this high-value set and assume reward for non-high-value pages is zero.

The tuple $\langle S, A, P, r, \beta \rangle$ forms an infinite horizon MDP. We do not introduce an explicit discount factor, since the value of most website goals are not diminished by the number clicks required to reach the goal. For example, the sale of an item on Amazon.com after 8 clicks generates the same profit as the same item bought after 2 clicks. [1] The exit state s_{exit} is absorbing and encountered from each state with non-zero probability, ensuring that infinite reward cannot be accumulated.

Policies

Given a fully specified reward function we wish to find a policy $\pi : S \rightarrow A$ that selects a version for each non-static webpage requested and maximizes the total expected reward EV^{π}:

$$\begin{aligned} V^{\pi}(s) &= r(s) + \sum_{s'} \Pr(s' \mid s, a) V^{\pi}(s') \\ EV^{\pi} &= V^{\pi}(s_{enter}) \end{aligned} \tag{7.1}$$

Note that the policy does not use information beyond the identity of the webpage requested; thus we can view the policy as determining a new static website.

Occupancy Frequencies

A policy induces a set of *occupancy frequencies* f. Recall that occupancy frequencies form a deterministic policy π are defined by the following constraints (see Equation 2.11):

$$f(s, \pi(s)) - \sum_{s'} \Pr(s \mid s', \pi(s')) f(s', \pi(s')) = \beta(s) \qquad \forall s \in S$$

[1]There may be other reasons to prefer a buyer reach a goal with fewer clicks. Generally such reasons can be modeled by additions to the state space without requiring an explicit discount factor.

In our domain occupancy frequencies have intuitive form, roughly capturing the proportional number of visits to a webpage.

Uncertain Reward

To allow for reward to be incrementally specified, we adopt our elicitation framework and use an imprecise specification of reward $\mathcal{R} \equiv \{\mathbf{r} \mid \mathbf{Cr} \leq \mathbf{d}\}$ defined by a set of linear constraints (with coefficients $\mathbf{C}, \mathbf{d}$), resulting in the imprecise reward MDP $\langle S, A, \Pr, \beta, \mathcal{R} \rangle$.

In the website optimization domain, linear constraints on the feasible set of reward functions can be derived from upper and lower bounds on the value of each goal page. Comparisons among goal pages can also be used. For instance there may be a primary goal s (e.g., item purchase) that is more valuable than any other goal s', leading to the constraint $r(s) \geq r(s')$. In the case of LinkedIn, the website design may be able to state upfront that the goal of a user upgrading to a paid account is more valuable than the user "sharing an update," or adding contacts. Note that while a website designer may be able to quickly communicate a preference ordering that ranks the set of goals, in most cases elicitation is still required to define the *strength* of preference, in order to precisely determine the necessary trade-offs between these goals.

Given an IRMDP of in the form described above, our goal is to effectively query the website designer in order to find a policy (representing an optimized website) with an acceptable regret with respect to the uncertainty over the value of goal pages. In the next section we discuss how we generate IRMDPs so that we may empirically investigate the effectiveness of our approach in this domain.

7.3 Models For Empirical Analysis

We are not aware of existing public datasets that capture the results of website version tests; however, there are a number of public datasets that record web traffic on static websites that can used as a foundation for *simulated* version-testing.

Next we detail a procedure for generating version-testing IRMDPs that involves: 1) specifying the probabilities P_{static} that correspond to actual measured web traffic on a static website, 2) specifying the set S_{vt} of pages to be version tested, 3) specifying the impact of each version on web traffic, 4) specifying the uncertain reward function w.r.t. a subset of “goal” pages.

7.3.1 Creating a Static Website with Traffic

The ACM SIGCOMM Internet Traffic Archive (Archive) is a repository that provides traces of Internet network traffic. Table 7.1 describes the details of four datasets from the Archive that we use to construct static websites. The datasets are derived from the HTTP access logs of: 1) the NASA Kennedy Space Center webserver, 2) the University of Calgary Department of Computer Science webserver, 3) University of Saskatchewan’s university-wide webserver, and 4) the FIFA World Cup 98 webserver.

Website	Website Size	Log Size
NASA	1,597	3,461,612
University of Calgary	4,757	726,739
University of Saskatchewan	6,078	2,408,625
World Cup 98	11,411	1,352,804,107

Table 7.1: HTTP Access Log details

We use each dataset to construct a model representing the probability $P_{static}(s'|s)$ by conducting counts of the instances in which a user clicks on a link to page s' from page s. Details of how this counting is performed can be found in Appendix D.1.

7.3.2 Simulating Webpage version-testing

For our purposes there are two significant short-comings in the public datasets: 1) they do not measure the impact of version-testing on various webpages, and 2) they do not specify reward-bearing “goal” pages. Next we propose methods for constructing a hypothetical model of both of these missing pieces; we begin with the selection of the “goal” pages and the specification of the uncertain reward polytope $\mathcal{R}$.

Specifying Reward Structure

One reasonable approach to specifying goal pages is to simply browse each website and speculate on pages that the designers would most prefer users to visit. Unfortunately, the datasets do not represent the websites in a browsable form;[1] instead we select the set G of goal pages based on link structure and measured user traffic. Following common assumptions (Hollink, van Someren, and Wielinga, 2007; Perkowitz and Etzioni, 1997; Rupert, Rattrout, and Hassas, 2008; Wang, Wang, and Ip, 2006; Zhou, Chen, Shi, Zhang, and Wu, 2001) we choose a set of goal pages that are both popular and are encountered near the end of a website navigation session (i.e., the page is the last page visited before the user exits the site).

We generate a true (but hidden) reward function which reflects the as-yet-unspecified preferences of the website designer. In lieu of a fully specified reward function, we begin with a reward polytope representing our uncertainty about the true reward function in the form of linear constraints. We assume that the website designer has predetermined the set of goal pages and that the reward for non-goal pages is zero; formally, for all $s \in S \setminus G$ we assume that: $r(s) = 0, \ \forall\, r \in \mathcal{R}$. Thus the dimension of $\mathcal{R}$ is effectively $|G|$.

A simple starting point for the simulation of the true reward function and reward polytope is to uniformly sample the value of the true reward function and to bound each unknown reward parameter by a minimum and maximum possible reward value. For instance given a minimum and maximum reward value of 0.0 and 1.0 respectively, we define the reward polytope: $\mathcal{R} \equiv \{\, r \mid 0.0 \leq r(s) \leq 1.0 \ \ \forall\, s \in G \,\}$.

Along with specifying the goal pages, website designers are often able to place simple bounds on reward a priori through a partial ordering of the goal pages in terms of importance. For instance, an e-commerce website has many different goals beyond simply leading a customer to immediately purchase an item. While the website designer cannot easily assess the exact tradeoffs—between, for example, a user purchasing an item, creating an account with

[1]The websites are no longer hosted online in their recorded form, and the datasets do not contain the actual HTML of each site.

credit card information, leaving a review—the designer may be able to give a partial ranking, stating that: a purchase preferred to creating account and creating an account is preferred to leaving review. We can directly incorporate such ordering constraints into the uncertain reward polytope. Given an ordered pair of goal pages: $s_a \succ s_b$, $\mathcal{R} \equiv \{\ r \mid 0.0 \leq r(s) \leq 1.0\ \ \forall\ s \in G$ and $r(s_a) \geq r(s_b)\ \}$. For our experiments, we omit any reward goal page ranking constraints and show effective elicitation can be achieved with a minimal amount of reward information. Adding goal page ranking results would only improve the results presented below.

Simulating Version-Testing

Generally, the purpose of testing a new version of a page is to boost traffic toward a specific goal. We can break down our method for simulating version-testing into the following steps: we select a set of pages to version test; then for each version of each page we select a goal page as the *version target*; finally, we simulate how each version shapes traffic toward the target, increasing traffic toward the target and decreasing traffic to other pages. The full details of each of these steps can be found in Appendix D.2.

7.4 Reducing Website MDPs

Any page on which version-testing was not carried out corresponds to a state with a single static action. If such a page is not a reward-bearing goal page, we safely eliminate it and reduce the size of MDP. In practice the number of pages that are neither goal pages nor pages being version tested constitutes a large fraction of the website; eliminating this fraction can significantly reduce the dimension of the corresponding MDP and positively impact the efficiency of our computational methods.

For generality, we assume the presence of a discount factor in the following discussion; the results can be directly applied to our version testing domain by setting $\gamma = 1$. We define a state to be *redundant* if the state bears no reward, is not a starting state, and there is a single default action that may be taken in the state. More formally a state y is redundant iff $r(s, a_0) = 0$ and $\beta(y) = 0$ and $A(y) = \{a_0\}$. We seek to create a reduced MDP in which the redundant state is eliminated. Once we establish the transformation w.r.t. to a single state, extending the transformation to a larger set of states is straightforward.

To eliminate the state y, we update the transitions of all states $s, s' \neq y$ for which the transition $s \to y \to s'$ has non-zero probability. Recall that we omit the static action a_0 from the probability of transitioning from a non-version page y: $P(s|y) = P(s|y, a_0)$ for all $s \in S$. We create a new transition function $P_{\text{-}y}(s'|s, a)$ to reflect the discounted probability accrued from transitions through state y:

$$\begin{aligned}
P_{\text{-}y}(s'|s,a) = {} & P(s'|s,a) \\
& + P(s'|y)\gamma P(y|s,a) \\
& + P(s'|y)\gamma P(y|y)\gamma P(y|s,a) \\
& + P(s'|y)\gamma P(y|y)\gamma P(y|y)\gamma P(y|s,a) \\
& + \cdots \\
= {} & P(s'|s,a) + P(s'|y)\gamma P(y|s,a)\left[\sum_{t=0}^{\infty}\Big(\gamma P(y|y)\Big)^{t}\right] \\
= {} & P(s'|s,a) + P(y|s,a)\gamma P(s'|y)\frac{1}{1-\gamma P(y|y)} \qquad (7.2)
\end{aligned}$$

Given an underlying MDP and a redundant state y. Let $MDP_{\text{-}y} = \langle S_{\text{-}y}, A, P_{\text{-}y}, \gamma, r\rangle$ be our reduced MDP with $S_{\text{-}y} = S \setminus \{y\}$ and $P_{\text{-}y}$ defined as in Equation (7.2). Given a policy π w.r.t. to the original MDP, let $\pi_{\text{-}y}$ define the policy that omits of the redundant states eliminated from our reduced MDP. Formally: $\pi_{\text{-}y} = \{\pi_{\text{-}y} {=} \pi(s) \mid \forall s \in S \setminus \{y\}\}$

Theorem 3 *Let $V^{\pi_{\text{-}y}}$ be the expected value of policy $\pi_{\text{-}y}$ w.r.t. to the reduced $MDP_{\text{-}y}$ and let V^{π} be the expected value of policy π w.r.t. to the original MDP. Then $V^{\pi_{\text{-}y}}(s) = V^{\pi}(s)$ $\forall s \in S \setminus \{y\}$.*

Proof of this theorem can be found in Appendix A.4. This result allows us to safely remove a redundant state y creating a without altering the value function of a policy. Thus an optimal policy w.r.t. to the original MDP will remain optimal in the reduced MDP. The result extends to our minimax regret calculations since minimax regret is defined in terms of value functions. Minimax regret can be alternately expressed as follows:

$$MMR(\mathcal{R}) = \min_{\pi \in \mathcal{P}} \max_{\pi' \in \mathcal{P}} \min_{\mathbf{r} \in \mathcal{R}} \quad V_{\mathbf{r}}^{\pi'} - V_{\mathbf{r}}^{\pi},$$

where $V_{\mathbf{r}}^{\pi} = \mathbf{f}^{\pi} \cdot \mathbf{r}$, and $\mathcal{P}$ is the set of feasible policies.

In our website optimization domain, Theorem 3 implies that we may create a new MDP with states S' and transition function P' that removes *all* redundant pages (i.e., non-goal pages that are not being version tested) by simply reapplying this process to all redundant pages in turn. This process is detailed in Algorithm 7.

Algorithm 7: Redundant State Elimination Algorithm

```
Input:
  $\langle S, A, P, \gamma, r, \beta \rangle \leftarrow$ The underlying MDP
  $Y \leftarrow$ a set of redundant states to be eliminated
foreach $y \in Y$ do
    $S \leftarrow S - y$
    foreach $s \in S, s' \in S, a \in A$ do
        $P(s'|s,a) \leftarrow P(s'|s,a)P(s'|s,a) + P(y|s,a)\gamma P(s'|y,a_0)\frac{1}{1-\gamma P(y|y,a_0)}$
    end
end
Return: $\langle S, A, P, \gamma, r, \beta \rangle$
```

There is related work on *macro-actions* (Hauskrecht, Meuleau, Kaelbling, Dean, and Boutilier, 1998), which are also referred to as *options* (Sutton, Precup, and Singh, 1998). Macro-actions represent local policies that dictate a course of action for a number of steps; they naturally capture the consequence of entering a redundant state in our website MDP where a default action is executed until a transition away from the redundant state. Hauskrecht et al. (1998) detail a hierarchical approach (using an abstract MDP) that works with macro-actions that would serve to abstract away redundant states. Parr (1998) presents a related approach to decompose MDPs into a set of weakly-coupled local MDPs. Each local MDP is solved to construct a *cache* of policies which are non-dominated given the possible behaviour of *adjacent* local MDPs. Each cache can can be viewed as a set of macro-actions for in the global MDP.

7.5 Experiments

7.5.1 Setup

We assess our approach on the four datasets mentioned above. We select 10 goal pages and simulate the testing of 5 different versions of 50 different pages for each dataset. The impact of each version increases traffic by an average of 5% (drawn from N$(0.05, 0.025)$) towards the version target. The random generation of version-testing IRMDPs with these parameters was repeated for each dataset over 20 runs and all results were averaged. We perform the reduction describe in Section 7.4; the resulting IRMDP has:

$$|S| = |\text{version tested pages}| + |\text{goal pages}| + |\text{exit/entry pages}| = 50 + 10 + 2 = 62$$
$$|A| = |\text{versions}| = 5$$
$$|\mathcal{R}| = |\text{goals}| = 10$$

7.5.2 Efficiency

We begin by pre-computing the set of nondominated policies (using our NRV algorithm). Table 7.2 shows results related to computational efficiency. First observe the number of non-dominated policies (denoted $|\Gamma|$) is a small fraction of the 5^{62} feasible policies. The set Γ of nondominated policies is small enough to enable online interactive elicitation; Table 7.2 shows that our procedure for computing minimax regret using Γ takes less than a second. Furthermore, the setwise max regret computation (using Γ) detailed in Section 4.3 is quick enough to enable the myopically optimal online selection of full policy queries.

	Offline		Online	
Dataset	**NRV-time**	$\lvert\Gamma\rvert$	**MMR-time**	**SMR-time**
NASA	1826	225.7	0.198	1.173
University of Calgary	3592	380.1	0.312	4.019
University of Saskatchewan	2321	245.7	0.270	3.106
World Cup 98	4023	394.6	0.504	6.732

Table 7.2: Average computation time in seconds of nondominated policies (NRV-time) along with the average number of nondominated policies generated ($|\Gamma|$), and computation time for minimax regret (MMR-time) and optimal policy queries using setwise max regret (SMR-time).

We undertake elicitation using two approaches: the first selects bound queries using the current solution heuristic; the second selects myopically optimal full policy queries. Details of both approaches can be found in Chapter 4.

7.5.3 Lift

Figure 7.3 shows (unsurprisingly) that myopically optimal full policy selection provides for more effective query selection on all datasets (compared to heuristically selected bound queries).

Table 7.3 summarizes some measures of the improvement or "lift" that we observe with the myopically optimal query selection approach. As a baseline we compute *EV-static*, the expected value of the initial (pre-version-tested) static website w.r.t. true reward. After elicitation has reduced minimax regret to zero, the minimax regret optimal policy is provably the optimal policy w.r.t. to true reward. We denote the expected value of this optimal policy as *EV-opt* and use the phrase *full lift* to quantify the improvement: full lift = (EV-opt − EV-static). In our experiments it takes an average of 11–24 queries (across all datasets) to reduce regret to zero, producing full lift from 15–38 percent. Next, we measure the (ex-post) *initial lift* as (EV-opt − Initial MMR). Intuitively, this provides bound on the improvement offered by the initial minimax regret optimal policy. We observe after the fact that the initial minimax regret policy provides an average improvement of at least 2–24 percent over the intial static website (across the datasets) and that by completing reward elicitation (reducing regret to zero) we increase this improvement to 15–38 percent.

Dataset	EV-static	EV-opt	Initial MMR	Initial lift	Full lift
NASA	1.0	1.151	0.126	0.023	0.151
University of Calgary	1.0	1.381	0.212	0.168	0.381
University of Saskatchewan	1.0	1.223	0.170	0.106	0.223
World Cup 98	1.0	1.325	0.077	0.248	0.325

Table 7.3: EV-static measures the expected value of the static (pre-version-test) website (w.r.t. true reward). All values in the table are normalized w.r.t. to EV-static. EV-opt measures the expected value of the optimal (post-version-test) policy (w.r.t. true reward). Initial MMR measures the initial minimax regret of the IRMDP. Initial lift is (EV-opt - Initial MMR) and constitutes an ex-post lower bound on the improvement over the static policy that the mmr optimal policy yields. Full lift measures the improvement that the optimal policy (found when MMR is reduced to zero) offers over the static policy.

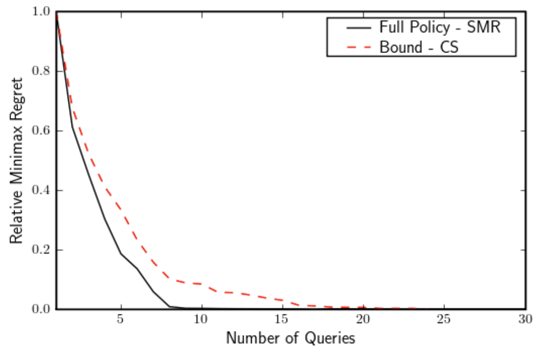
7.5.4 Elicitation Effectiveness

Figure 7.3 shows the results of elicitation on each dataset (averaged over 20 runs). For each dataset minimax regret is quickly reduced. For each website and strategy regret is reduced to half its original value in less than 5 queries, and regret is reduced to a fifth of its original value in less than 10 queries. Regret is reduced to zero in all cases in under 30 queries, and in some cases (NASA with Full Policy SMR) regret is reduced to zero in just 10 queries—one query per reward parameter. In general full policy queries selected using setwise max regret reduce minimax regret more quickly than bound queries selected using the current solution heuristic. This should be unsurprising since the response to full policy queries contain more information about the reward function. We next discuss how full policy queries may be posed to a website designer in a cognitively reasonable manner.

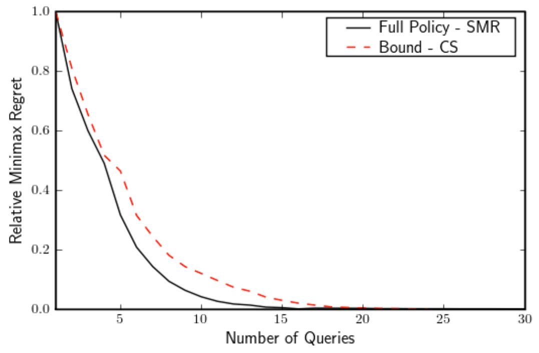
7.5.5 Full Policy Queries

Let f and f' be the occupancy frequencies induced by the two policies to be compared in a full policy query. We can summarize each policy by examining its impact on traffic to a goal page s_g by examining $f(s_g, a_0)$ which represents the proportion of time a user will visit goal page s_g (since goal pages are not version tested, each goal page has default action a_0).

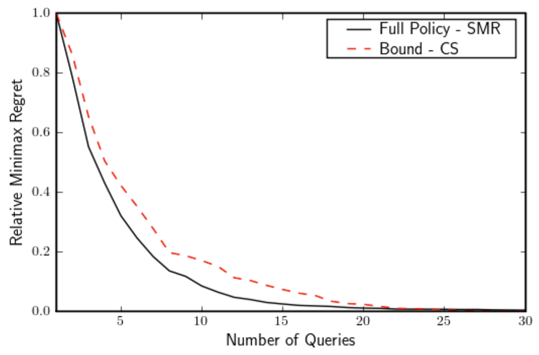
Figure 7.4 shows two possible representations for the goal page occupancy frequencies.



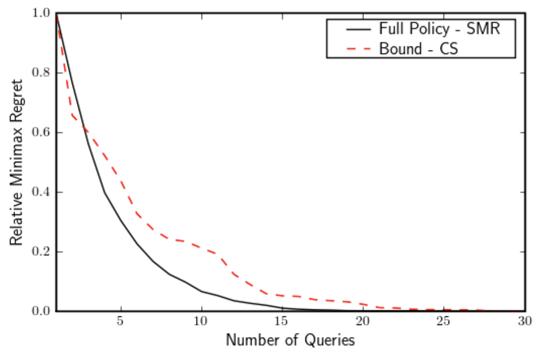
(a) NASA



(b) University of Calgary



(c) University of Saskatchewan



(d) World Cup

Figure 7.3: Results of reward elicitation: relative minimax regret vs. query number.

The first row of the figure shows the normalized occupancy frequencies for each goal page (labeled A through H). The second row shows the "delta" between each occupancy frequencies of each policy and allows the designer to focus on the relative trade-offs presented by each policy. Note the most significant trade-offs in query #1 are among the first four goal pages. Ignoring the small differences among the last four goal pages, the designer roughly needs to decide if increased traffic to goal page B is worth a decrease in traffic to pages A, C, and D.

Of course one may resort to simpler bound queries that do not require simultaneously reasoning about all goals. The results shown in Figure 7.3 suggest that using bound queries rather than myopically optimal policy queries will have a very slight negative impact elicitation effectiveness; for instance, on the NASA dataset (7.3 (a)) an additional 12 queries are required to reduce regret to zero.

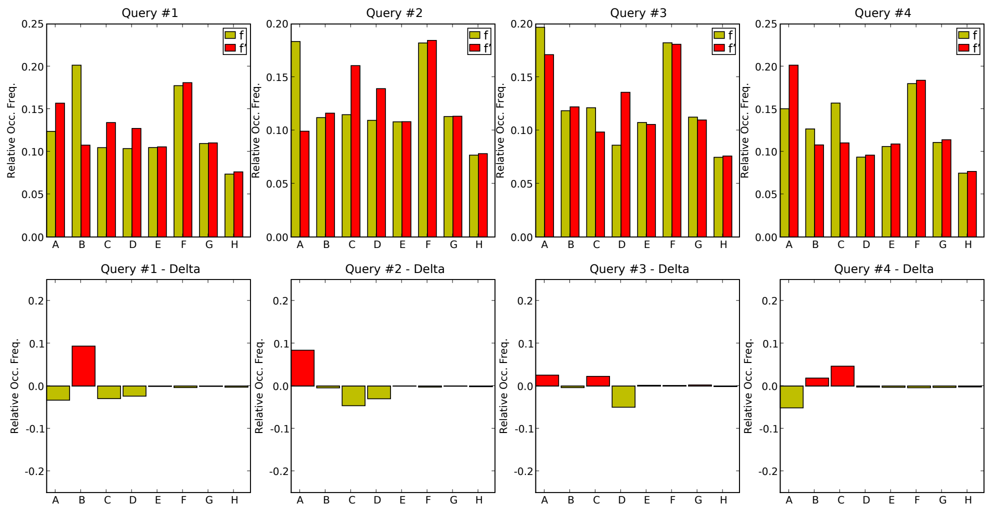


Figure 7.4: Representations for queries comparing goal page occupancy frequencies.

7.6 Summary and Conclusions

Given a sequence of Web pages accessed by a user, existing work has tackled the problem of learning to predict subsequent Web page requests using various high-order Markov models. (Deshpande and Karypis, 2004; Hassan, Jones, and Klinkner, 2010; Sarukkai, 2000). The work suggests many uses for these predictive user models such as improving web cache performance and personalizing websites; however, a principled decision theoretic model for actions based on prediction is not considered in prior work. Richer models of user behaviour could be incorporated into our approach at the cost of increasing the complexity of the IRMDP state state and transition model.

Archak et al. (2010) develop a model for online advertising that captures the behavior of users, including their responses to advertising actions, as a Markov chain. This allows for the computation of the optimal policy for allocating budget to a specific user in a way that adapts the allocation based on observed behavior using a constrained Markov decision process (Nikolay Archak and Muthukrishnan, 2010). This domain avoids the difficulties of reward specification, since, there is a simple easily observable reward function that measures a single "conversion" goal.

Existing work on adaptive websites takes the perspective of improving user experience. User goals are inferred from web access logs and structural changes (e.g., adding new links or emphasizing existing links) are suggested using several heuristic approaches that aim to improve various user-centric objectives such as minimizing the path to frequently accessed content (Hollink et al., 2007; Perkowitz and Etzioni, 1997; Rupert et al., 2008; Wang et al., 2006; Zhou et al., 2001). Version testing will implicitly capture the goals of users; however, our work optimizes the goals of the website *designer* (which may not align with the goals of the user) and performs the optimization in a sound decision theoretic manner.

In this chapter we described how website optimization through version-testing may be modeled as an MDP. Each choice of versions distributes the expected flow of website traffic among different (often competing) goals. A reward function capturing the preferences of the website designer must precisely specify the relative value of these goals—a cognitively demanding and time consuming task. Our reward elicitation approach lowers the burden of reward specification in this domain, enabling website designers to optimize their websites without undertaking the task of full reward specification.

To demonstrate the effectiveness of our approach in this domain we generated IRMDPs that simulate version-testing on a set of existing websites. Each simulation is built upon a large public dataset capturing a model of user behaviour on a static website. We show that the resulting IRMDPs may be reduced to a fraction of their original size by eliminating redundant states; exact minimax regret computation and myopically optimal query selection are efficient enough on the resulting IRMDP to enable elicitation with real-time user interaction.

We described how full policy queries in this domain may be communicated so as to reduce their complexity and lower the cognitive burden on users. The results of our elicitation experiments using these queries suggests that using few queries per goal we may identify the optimal policy determining the optimal selection of page versions, and that this optimal policy will significantly increase the expected reward.

7.6.1 Contributions

Completed work from this section constitutes the following contributions to the literature:

- The empirical analysis of public datasets demonstrating effective version-testing optimization on websites with thousands of pages.
- Examples of effective and cognitively reasonable full policy queries

Chapter 8

Conclusions and Discussion

In this thesis we have developed a framework for specifying user preferences over the operation of complex systems for reasoning under uncertainty.

Specifically, we focus on Markov decision processes (MDPs), a well-established model for *sequential* reasoning under uncertainty that requires the specification of both a model of the stochastic dynamics and a reward function capturing user preferences. Many real-world reasoning problems are naturally modeled as MDPs. For instance, planning and logistics at large organisations often involve optimizing processes with series of actions with stochastic effects. In this thesis we described three concrete examples and detailed MDPs for website optimization, autonomic computing, and cognitive assistance systems. While dynamics can be learned by observation of the environment, the reward function reflects the subjective preferences of some user and can require sophisticated human judgement to assess relevant tradeoffs.

Our framework extends well understood decision-theoretic approaches representing and eliciting human preferences (Keeney and Raiffa, 1976) to sequential decision problems. We make no assumptions about prior probabilistic information about preferences; instead we adopt a *strict uncertainty* model and quantify the unknown user preferences in terms of a set of feasible reward functions.

A significant feature of our framework is the facility for *incremental* elicitation. At each step of elicitation, we generate a robust policy that minimizes the worst-case loss (i.e., minimax regret) w.r.t. to the remaining uncertainty over reward. If the bound on loss is too high, a number of different queries may be posed to the user

By supporting decision-theoretically sound elicitation with effective computation, we demonstrate the potential for practical elicitation of reward functions; thereby removing a significant barrier to the specification (and use of) models for sequential decision making under uncertainty.

8.1 Summary of Results

Computing Robust Policies using Minimax Regret Chapter 3 focuses on computing minimax regret. We describe an exact constraint generation algorithm; however, the mixed integer programming at the heart of the approach forms a computational bottleneck, limiting its usefulness to small IRMDPs. Given the theoretical intractability of minimax regret for IRMDPs (Xu and Mannor, 2009) this limitation is unsurprising.

We developed several computationally efficient approximations. Empirically, an alternating optimization approximation offers low error and can be used to inform query selection, however, it does not yield an upper bound on minimax regret that could provide a guarantee to the user. The reformulation-linearization (RLT) approximation offers the desired upper bound, but empirically exhibits significantly higher error.

Next, we examined how a precomputed set Γ of nondominated policies can be used to compute minimax regret. To generate the set of nondominated policies, we began by describing the πWitness algorithm. The complexity of both nondominated policy generation and minimax regret computation is tightly tied to the cardinality of Γ which we observe to be related to the dimension of the reward function. Thus, IRMDPs with large state spaces and compact reward functions can be tackled efficiently; however, as reward dimensionality grows, an exact representation of Γ becomes less useful. We suggest an anytime variant of the πWitness algorithm for approaches for generating approximate sets Γ to approximate minimax regret; however, unlike our NRV algorithm this approach yields no bound on approximation error.

Reward Elicitation We introduced our approach reward elicitation for general IRMDPs (that may not exhibit factored structure) and detailed two heuristic approaches to selecting simple bound queries; the first, HLG, using information about only the feasible reward set; the second supplementing this with information from the current minimax regret solution.

Our empirical analysis suggests that current solution heuristic is far more effective than using only reward information; and further that using the current solution of minimax regret is more effective than other robust criteria (such as maxamin value). Elicitation with the current solution heuristic identified provably optimal policies (i.e., reducing regret to zero) using a small number of queries per reward parameter.

To complement our heuristic approach to selecting bound queries, we constructed a method for the optimal selection using more complex full policy queries. To enable the approach we proved that we may use setwise max regret (SMR) to find myopically optimal queries. This substitute measure is more amenable to constrained optimization and allows for more tractable computation. We detailed an exact computational approach using mixed integer programming and described how nondominated policies may be leveraged to reduce the computation to a series of linear programs.

Myopically optimal selection of full policy queries further improves elicitation effectiveness at the cost of additional computational overhead and cognitive burden; however, these costs can be manageable—the website optimization domain described in Chapter 7 is an example of a setting where a compact set of nondominated policies allows for tractable computation while a small number of reward bearing states enables the cognitively reasonable assessment full policy queries.

Leveraging Reward Structure We described how reward structure can be leveraged to improve the effectiveness of elicitation. We extend our approaches to compute minimax regret for IRMDPs with additive reward composed of local value function and develop decision-theoretically sound heuristics to elicit information about these local value functions and along with parameters specifying their global calibration.

We developed an exact approach to computing minimax regret in this setting. In the general case, we propose an exact method that uses a series of linear and mixed integer quadratic programs and we identify a special (though common) case in which the mixed integer program may be linearized. We provide tools for efficient approximation of minimax regret in this context by extending the alternating optimization and the relaxation-linearization technique (RLT) to handle additive reward structure with local value functions.

To demonstrate the effectiveness of our approach to eliciting additive rewards, we undertook an experimental assessment in two example domains. In each domain we observed that leveraging reward structure led to far more effective elicitation. In comparison, variants of each domain with unstructured reward: 1) required more queries; and 2) required more demanding full-state queries.

Online Minimax Regret Computation We revisited the online use of nondominated policies. Shrewd management of an approximate set of nondominated policies during elicitation can yield efficient computation of approximate minimax regret. We develop the NRV algorithm for anytime nondominated policy generation, which yields a bound on error and operates by generating policies to maximally reduce that error. Paired with our online management of nondominated policies, the NRV algorithm allows for approximate minimax regret computation with bounded error that is reduced as elicitation proceeds.

An empirical analysis on the COACH domain demonstrated we may quickly reduce error, probably reducing regret to zero (and identifying the optimal policy) using only only a small fraction of all nondominated policies. The online adjustment of nondominated policies when paired with the NRV algorithm removes a significant computational barrier to online reward elicitation for MDPs.

Reward Elicitation of Website Optimization Chapter 7 applies our reward elicitation framework to a website optimization problem. A reward function capturing the preferences of the website designer must precisely specify the relative value of these goals—a cognitively demanding and time consuming task. We described how website optimization through version testing may be modeled as an IRMDP.

We demonstrate the effectiveness of our approach in this domain by simulating version testing on a set of existing websites, where the simulation is built upon a large public dataset capturing a model of user behaviour on a static website. We show that the resulting IRMDPs may be reduced to a fraction of their original size by eliminating redundant states; exact minimax regret computation and myopically optimal query selection are efficient enough on the resulting IRMDP to enable elicitation with real-time user interaction.

We described how full policy queries in this domain may be communicated so as to reduce their complexity and lower the cognitive burden on users. The results of our elicitation experiments using these queries suggests that using few queries per goal we may identify the optimal policy determining the optimal selection of page versions, and that this optimal policy will significantly increase the expected reward.

We summarize our contributions in point form below. We include citations for work that has been previously published.

- An exact procedure for MMR computation using constraint generation and mixed integer programming (Regan and Boutilier, 2008, 2009)
- Several approximate methods for efficiently generating lower & upper bounds on MMR (Regan and Boutilier, 2008, 2009, 2011a)
- An exact approach to minimax regret computation leveraging nondominated policies (Regan and Boutilier, 2010)
- A polynomial algorithm (πWitness) for generating nondominated policies (Regan and Boutilier, 2010)
- The development of of volumetric and current solution heuristics for query selection during the elicitation of IRMDPs (Regan and Boutilier, 2009)
- A Proof of equivalence between min setwise max regret (SMR) and min worst-case regret (WR) for full policy queries
- A method for myopically optimal full-policy query selection based on SMR computation
- Exact and approximation algorithms computing minimax regret for IRMDPs with additive reward structure (Regan and Boutilier, 2010, 2011a)
- Decision-theoretically sounds heuristics for eliciting additive reward using local value functions (Regan and Boutilier, 2011a)
- The NRV algorithm for generating nondominated policies which provides an anytime bound on approximation error for minimax regret (Regan and Boutilier, 2011b)
- A method for adjusting the set of nondominated policies online, speeding up computation and improving the quality of approximation (Regan and Boutilier, 2011b)
- The empirical analysis of public datasets demonstrating effective version testing optimization on websites with thousands of pages.
- Examples of effective and cognitively reasonable full policy queries

8.2 Future Directions

Interface Design and User Studies The work described in this thesis has addressed the computational considerations of our reward elicitation framework while leaving some psychological considerations as future work. A practical real-world system for reward elicitation must take care to express queries in a form that is quickly understood by the user. We consider the user interface design required for such a system to be outside the scope of this work; however, we also consider it to be an important and vital aspect of building a system that is easy to use.

An example of how an effective user interface can be built around an elicitation system with queries similar to those suggested in this thesis can be found in UTPref System (Braziunas and Boutilier, 2010). The work also serves as an example of how to conduct a user study to empirically assess the effectiveness of the proposed elicitation approach with real users.

Indifference We currently impose no boundary on the precision with which a user may be expected to answer. Elicitation with bound queries tends to focus on a small set of high impact reward points repeatedly querying to slice the interval of feasible reward ever smaller. It is reasonable to assume that a user may have a limit to the precision with which they can answer these repeated queries. This motivates allowing users to answer queries with "I dont know". The implications of this response varies with query type. For bound queries we can assume that an "I dont know" response indicates that the true reward value is sufficiently close to the bound to render a decision difficult. Such a response constrains $r(s, a)$ to be within a small constant δ of the bound b. For comparison queries, the response suggests that the user is approximately indifferent between to the two options. For instance, indifference when comparing two policies, π and π', imposes the constraint: $|V^{\pi} - V^{\pi'}| \leq \delta$. Incorporating indifference requires minor changes in implementation and results in query response options which are more natural for a user and potentially increase the effectiveness of elicitation.

Leveraging Nondominated Policies with Structured Reward Our work incorporating additive reward with local value functions necessitated a quadratic reward parameterization: unknown local value function are multiplied by unknown scaling constants. It would be useful to extend our techniques for leveraging nondominated policies to this reward representation. However this involves identifying policies that are nondominated w.r.t. to both scaling function *and* local value function uncertainty.

The approach used by our NRV algorithm to enumerate the vertices of nondominated reward regions will not easily extend to regions which are defined by quadratic rather linear constraints. The πWitness algorithm is similar complicated, since finding witness reward points becomes a quadratic optimization.

Of course these issues do not impact *calibrated* additive reward functions for which there is no uncertainty over scaling constants. The autonomic computing domain provides one such example of an uncalibrated reward function. This suggests an alternative approach of performing full elicitation of all scaling constants before computing minimax regret and engaging in incremental elicitation.

Incorporating Probabilistic Priors We have designed our elicitation framework to not require on the presence of probabilistic information about the unknown reward function. However, were such information available, it could be incorporated into our framework to guide query selection—while we continue to use strict bounds on reward to generate policies with guarantees on regret.

Given a distribution over potential query responses, we could apply Bayesian methods, and select the query that minimize the *expected* minimax regret of the response. This is a principled approach that is likely to effectively reduce minimax regret in practice, however, it initially additional *a priori* information (i.e., the distribution over query responses) and the inference the inference required to maintain the distribution after a query response may require significant computational overhead (Wang and Boutilier, 2003).

Extending the Elicitation Framework This thesis focuses on the *active* elicitation of reward asking the user to answer queries. There are other approaches to accumulating information that result in linear constraints on the reward polytope that can be incorporated into our framework. In particular, the tools of inverse reinforcement learning Ng and Russell (2000) may be adopted in some settings to observe optimal behaviour being demonstrated. For instance, human care-givers could be monitored to glean reward information in the cognitive assistance domain. Such demonstrations induces constraints on the reward function that may serve to define the initial reward polytope. Our reward elicitation framework to further refine our knowledge of the reward function until a suitable level of regret is reached.

Furthermore, it may be reasonable to use inverse reinforcement learning *during* elicitation. The information provided by the current solution to the minimax regret computation may suggest regions of state or action space for which an expert demonstration may be beneficial. In practice, effectively choosing between queries and demonstrations at each stage of elicitation requires an additional user cost model to capture the impact of the additional burden imposed by demonstrations.

Specifying Reward Structure The number of reward bearing states in real-world MDPs can be extremely small compared to the size of the state space. While a domain expert may be able to specify which state-action pairs should be the focus of elicitation, it can be desirable to leave the specification of structure of the reward function up to the user during elicitation. Recent work conducted by Boutilier, Regan and Viappiani (2009a; 2009b; 2010) has examined a similar scenario in the single-step decision making setting. They enable the user to specify groupings of individual features of the decision space to form composite features that are used during elicitation. For instance, elicitation can determine that one user has specific preferences about "safe cars" that are heavy, have high ground clearance, child restraints, side air-bags and anti-lock brakes. Another user may consider "safe" to include a high-performance suspension and roll-bars. In our website optimization domain, subjective features could be used to help define ambiguous goals such as “user engagement” that may vary across website designers. A significant component of this future work is the development of a variant of the optimization performed to compute minimax regret with respect to both uncertainty over utility and uncertainty with respect to the definition of composite features.

Appendix A

Proofs

A.1 Proof of Observation 1

Recall that from our definition of pairwise max regret (PMR):

$$\operatorname*{argmax}_{\mathbf{g}\in\mathcal{F}} PMR(\mathbf{f},\mathbf{g},\mathcal{R}) = \operatorname*{argmax}_{\mathbf{g}\in\mathcal{F}} \max_{\mathbf{r}\in\mathcal{R}} \quad \mathbf{g}\cdot\mathbf{r} - \mathbf{f}\cdot\mathbf{r}$$

Observation 1 *Given an IRMDP and policy* $\mathbf{f}$*:* $\operatorname{argmax}_{\mathbf{g}\in\mathcal{F}} PMR(\mathbf{f},\mathbf{g},\mathcal{R}) \in \Gamma$.

PROOF Assume the adversary policy g is not nondominated: $\mathbf{g} \notin \Gamma$. Then for all $\mathbf{r} \in \mathcal{R}$ there exists a $\mathbf{g}'$ such that $\mathbf{g}'\cdot\mathbf{r} > \mathbf{g}\cdot\mathbf{r}$ (otherwise for all $\mathbf{g}' \in \mathcal{F}$, it would be the case that $\mathbf{g}\cdot\mathbf{r} \geq \mathbf{g}'\cdot\mathbf{r}$, thus g would be nondominated). Let $\mathbf{r}'$ be the pairwise reward that maximizes $PMR(\mathbf{f},\mathbf{g},\mathcal{R})$. Since $\mathbf{g}'\cdot\mathbf{r}' > \mathbf{g}\cdot\mathbf{r}'$, the adversary policy g cannot be the pairwise regret maximizing policy and we have a contradiction.

□

A.2 Proof of Theorem 1 (Witness Theorem)

Theorem 1 *Let* $\Gamma' \subsetneq \Gamma$ *be a (strictly) partial set of nondominated policies. Then there is an* $\mathbf{f} \in \Gamma'$*, an* (s,a)*, and an* $\mathbf{r} \in \mathcal{R}$ *such that* $\mathbf{f}^{s:a}\cdot\mathbf{r} > \mathbf{f}'\cdot\mathbf{r}, \quad \forall\, \mathbf{f}' \in \Gamma'$.

PROOF We prove the result by contradiction and begin by assuming there exists no 4-tuple of $\mathbf{f}, s, a, \mathbf{r}$ such that such that $\mathbf{f}^{s:a} \cdot \mathbf{r} > \mathbf{f}' \cdot \mathbf{r}, \quad \forall\, \mathbf{f}' \in \Gamma'$.

1. Select $\mathbf{f}^* \in \Gamma \setminus \Gamma'$ and $\mathbf{r} \in \mathcal{R}$ such that $\mathbf{f}^* \cdot \mathbf{r} > \mathbf{f}' \cdot \mathbf{r}$ for all $\mathbf{f}' \in \Gamma'$. If such a pair $\mathbf{f}^*, \mathbf{r}$ did not exist then it would imply Γ' contained the full set of nondominated policies since for all $\mathbf{f}' \in \Gamma'$ it would be the case there exists a reward point $\mathbf{r}$ such that $\mathbf{f}' \cdot \mathbf{r} \geq \mathbf{f} \cdot \mathbf{r}$ for all $\mathbf{f} \in \mathcal{F}$.

2. Select the nondominated policy $\mathbf{f} \in \Gamma'$ that maximizes value at $\mathbf{r}$:

$$\mathbf{f} = \operatorname*{argmax}_{\mathbf{f} \in \Gamma'} \; \mathbf{f} \cdot \mathbf{r}$$

3. By construction, the policy $\mathbf{f}$ is not optimal for the given reward $\mathbf{r}$. We now show there is a local adjustment of $\mathbf{f}$ that improves value w.r.t. to the reward $\mathbf{r}$. For clarity we proceed using the traditional definition of a policy, $\pi : S \to A$ (rather than a policy as occupancy frequencies).

   Let $\pi_{\mathbf{f}}$ be the policy induced by $\mathbf{f}$. Since $\pi_{\mathbf{f}}$ is not optimal w.r.t. to reward $\mathbf{r}$, there is a state s at which we may improve the value of $\pi_{\mathbf{f}}$ by performing a Bellman backup (Puterman, 1994):

$$a = \operatorname*{argmax}_{a \in A} \; r(s,a) + \gamma \sum \gamma P(s'|s,a) V_{\mathbf{r}}^{\pi_{\mathbf{f}}}(s')$$

   Let $\pi_{\mathbf{f}}^{s:a}$ be the local adjustment that performs this improvement by taking action a for a single step in state s and following π thereafter. We have taken care to construct this local adjustment so as to offer improvement; thus it is the case that: $\boldsymbol{\beta} V_{\mathbf{r}}^{\pi^{s:a}} > \boldsymbol{\beta} V_{\mathbf{r}}^{\pi}$.

4. Recall that, as defined in Section 3.7.1, the local adjustment $\mathbf{f}^{s:a}$ w.r.t. a policy π is the set of occupancy frequencies that result from taking action a for one step and and following π thereafter. Thus $\mathbf{f}^{s:a}$ is the set of occupancy frequencies induced by $\pi^{s:a}$.

Since, $\beta V^{\pi^{s:a}} = \mathbf{f}^{s:a}\cdot\mathbf{r}$ and $\beta V^{\pi^{s:a}} > \beta V^{\pi}$, it must be the case that $\mathbf{f}^{s:a}\cdot\mathbf{r} > \mathbf{f}\cdot\mathbf{r}$ and we have constructed a tuple $\mathbf{f}, s, a, \mathbf{r}$ such that,

$$\mathbf{f}^{s:a}\cdot\mathbf{r} > \mathbf{f}\cdot\mathbf{r} \geq \mathbf{f}'\cdot\mathbf{r}, \quad \forall\, \mathbf{f}' \in \Gamma',$$

providing a contradiction, proving our theorem.

□

A.3 Proof of Theorem 2

The following proofs and lemmas closely follow the work of Boutilier and Viappiani (2009). We focus on comparison queries of the form $Z = \{\mathbf{f}_a, \mathbf{f}_b\}$; it is straightforward to extend the proofs to the case where $|Z| > 2$. Recall that:

$$\mathcal{R}^{Z\to\mathbf{f}_a} \equiv \big\{\mathbf{r} \in \mathcal{R} \mid \mathbf{f}_a\cdot\mathbf{r} \geq \mathbf{f}_b\cdot\mathbf{r}\big\}$$
$$WR(Z) = \max_{\mathbf{f}_i \in Z}\ MMR(\mathcal{R}^{Z\to\mathbf{f}_i})$$
$$SMR(Z) = \max_{\mathbf{f}_i \in Z}\ MR(\mathbf{f}_i, \mathcal{R}^{Z\to\mathbf{f}_i})$$

Lemma 1 *For any $Z \in \mathcal{Z}$, $WR(Z) \leq SMR(Z)$.*

PROOF It is the case that:

$$\forall\, \mathbf{f}_i \in Z \qquad \min_{\mathbf{f}\in\mathcal{F}} MR(\mathbf{f}, \mathcal{R}^{Z\to\mathbf{f}_i}) \leq MR(\mathbf{f}_i, \mathcal{R}^{Z\to\mathbf{f}_i})$$

It follows that $MMR(\mathcal{R}^{Z\to\mathbf{f}_i}) \leq MR(\mathbf{f}_i, \mathcal{R}^{Z\to\mathbf{f}_i})$ holds for all $\mathbf{f}_i$ and thus it holds for the maximum $\mathbf{f}_i$. Therefore $WR(Z) \leq SMR(Z)$.

□

We make use of a transformation on the query Z that does not increase setwise max regret. We define following MMR-transformation T:

$$T(Z) = \{\mathbf{f}'_a, \mathbf{f}'_b\} \qquad \text{such that} \quad \mathbf{f}'_i = \operatorname*{argmin}_{\mathbf{f}\in\mathcal{F}} MR(\mathbf{f}, \mathcal{R}^{Z\to\mathbf{f}_i}) \quad \text{for } i \in \{a, b\}$$

Lemma 2 *For any $Z \in \mathcal{Z}$, let $Z' = T(Z)$; then $SMR(Z') \leq WR(Z)$.*

PROOF Let I be the set indices corresponding to the policies in the query Z. We make use of set $\mathcal{R}^{Z\to\mathbf{f}_i} \cap \mathcal{R}^{Z'\to\mathbf{f}'_j}$: the partition of utility space where $\mathbf{f}_i$ is preferred from Z and where $\mathbf{f}'_j$ is preferred from Z'. We show that WR and SMR can be represented in terms of $\mathcal{R}^{Z\to\mathbf{f}_i} \cap \mathcal{R}^{Z'\to\mathbf{f}'_j}$ as:

$$WR(Z) = \max_{i\in I, j\in I} \; MR(\mathbf{f}'_i,\ \mathcal{R}^{Z\to\mathbf{f}_i} \cap \mathcal{R}^{Z'\to\mathbf{f}'_j}) \tag{A.1}$$

$$SMR(Z') = \max_{i\in I, j\in I} \; MR(\mathbf{f}'_j,\ \mathcal{R}^{Z\to\mathbf{f}_i} \cap \mathcal{R}^{Z'\to\mathbf{f}'_j}) \tag{A.2}$$

Expression (A.1) can be verified through the following steps:

$$\begin{aligned}
WR(Z) &= \max_{i\in I} \; MMR(\mathcal{R}^{Z\to\mathbf{f}_i}) \\
&= \max_{i\in I} \; MR(\mathbf{f}'_i,\ \mathcal{R}^{Z\to\mathbf{f}_i}) && \left[\textit{since } \mathbf{f}'_i = \operatorname*{argmin}_{\mathbf{f}\in\mathcal{F}} MR(\mathbf{f}, \mathcal{R}^{Z\to\mathbf{f}_i})\right] \\
&= \max_{i\in I} \max_{\mathbf{r}_{wr}\in\mathcal{R}^{Z\to\mathbf{f}_i}} \mathbf{f}^*_{\mathbf{r}_{wr}} \cdot \mathbf{r}_{wr} - \mathbf{f}'_i \cdot \mathbf{r}_{wr} && \left[\textit{by definition of MR}\right] \qquad \text{(A.3)} \\
&\qquad \text{where } \mathbf{f}^*_{\mathbf{r}_{wr}} = \operatorname*{argmax}_{\mathbf{f}\in\mathcal{F}} \mathbf{f} \cdot \mathbf{r}_{wr} \\
&= \max_{i\in I, j\in I} MR(\mathbf{f}'_i,\ \mathcal{R}^{Z\to\mathbf{f}_i} \cap \mathcal{R}^{Z'\to\mathbf{f}'_j}) && \left[\textit{since } \mathbf{r}_{wr} \in \mathcal{R}^{Z'\to\mathbf{f}'_j} \textit{ for some } j \in I\right] \qquad \text{(A.4)}
\end{aligned}$$

The regret maximizing reward $\mathbf{r}_{wr}$ from (A.3) will maximize the expression (A.4) since the expression selects the partition $\mathcal{R}^{Z'\to\mathbf{f}'_j}$ to maximize regret and $\mathbf{r}_{wr} \in \mathcal{R}^{Z'\to\mathbf{f}'_j}$ for some $j \in I$.

Expression (A.2) can be verified similarly:

$$\begin{aligned}
SMR(Z') &= \max_{j\in I} \;\; MR(\mathbf{f}'_j,\ \mathcal{R}^{Z'\to\mathbf{f}'_j}) && \Big[\textit{by definition of SMR}\Big]\\
&= \max_{j\in I} \;\; \max_{\mathbf{r}_{smr}\in\mathcal{R}^{Z'\to\mathbf{f}'_j}} \mathbf{f}^*_{\mathbf{r}_{smr}}\cdot\mathbf{r}_{smr} - \mathbf{f}'_j\cdot\mathbf{r}_{smr} && \Big[\textit{by definition of MR}\Big]\\
&\qquad\qquad \text{where } \mathbf{f}^*_{\mathbf{r}_{smr}} = \operatorname*{argmax}_{\mathbf{f}\in\mathcal{F}} \mathbf{f}\cdot\mathbf{r}_{smr}\\
&= \max_{i\in I, j\in I} MR(\mathbf{f}'_j,\ \mathcal{R}^{Z'\to\mathbf{f}'_j}\cap\mathcal{R}^{Z\to\mathbf{f}_i}) && \Big[\textit{since } \mathbf{r}_{smr}\in\mathcal{R}^{Z\to\mathbf{f}_i} \textit{ for some } i\in I\Big]
\end{aligned}$$

We now compare expressions (A.1) and (A.2) in terms of the component with i and the component with j. If $i = j$, then then the two MR components are the same. If $i \neq j$, consider any $\mathbf{r} \in \mathcal{R}^{Z\to\mathbf{f}_i} \cap \mathcal{R}^{Z'\to\mathbf{f}'_j}$. Since $\mathbf{r} \in \mathcal{R}^{Z'\to\mathbf{f}'_j}$, it must be the case that $\mathbf{f}'_j \cdot \mathbf{r} \geq \mathbf{f}'_i \cdot \mathbf{r}$. Therefore $MR(\mathbf{f}'_i, \mathcal{R}^{Z\to\mathbf{f}_i} \cap \mathcal{R}^{Z'\to\mathbf{f}'_j}) \geq MR(\mathbf{f}'_j, \mathcal{R}^{Z\to\mathbf{f}_i} \cap \mathcal{R}^{Z'\to\mathbf{f}'_j})$, and each element of the expression for SMR (A.2) is no greater than its correspondent in the WR expression (A.1). Thus $SMR(Z') \leq WR(Z)$.

□

Theorem 2 *The query $Z^* \in \mathcal{Z}$ that minimizes setwise max regret also minimizes worst-case regret. Formally,*

$$\textit{if} \quad Z^*_{smr} = \operatorname*{argmin}_{Z\in\mathcal{Z}} SMR(Z) \quad \textit{then} \quad WR(Z^*_{smr}) = \operatorname*{argmin}_{Z\in\mathcal{Z}} WR(Z).$$

PROOF We proceed by contradiction. Assume that Z^*_{smr} does not minimize worst case regret. Consequently there exists a $Z \in \mathcal{Z}$ such that $WR(Z) < WR(Z^*_{smr})$. We apply our transformation T yielding $Z' = T(Z)$. Observe,

$$\begin{aligned}
WR(Z^*_{smr}) &\leq SMR(Z^*_{smr}) \quad [\text{ by Lemma 1 }]\\
SMR(Z') \leq WR(Z) &< WR(Z^*_{smr}) \quad [\text{ by Lemma 2 }]
\end{aligned}$$

This implies $SMR(Z') < SMR(Z^*_{smr})$, contradicting the optimality of Z^*_{smr} w.r.t. SMR.

□

A.4 Proof of Theorem 3

In Section 7.4 we defined a state y to be *redundant* iff for all states s (and with default action a_0): $r(s, a_0) = 0$ and $\beta(y) = 0$ and $A(y) = \{a_0\}$. We defined a transition function $P_{\text{-}y}$ that accounted for the removal of a redundant state y:

$$P_{\text{-}y}(s'|s,a) = P(s'|s,a) + P(y|s,a)\gamma P(s'|y)\frac{1}{1-\gamma P(y|y)} \tag{A.5}$$

Note that we use the more concise notation $P(s|y)$ to express the probability of transitioning from a redundant state y, given the implied default action a_0 to subsequent state s.

Given an underlying MDP and a redundant state y, let $MDP_{\text{-}y} = \langle S_{\text{-}y}, A, P_{\text{-}y}, \gamma, r\rangle$ be our reduced MDP with $S_{\text{-}y} = S \setminus \{y\}$ and $P_{\text{-}y}$ defined as in Equation (7.2). Given a policy π w.r.t. to the original MDP, let $\pi_{\text{-}y}$ define the policy that omits mention of the redundant states eliminated from our reduced MDP. Formally: $\pi_{\text{-}y} = \{\pi_{\text{-}y} = \pi(s) \mid \forall s \in S \setminus \{y\}\}$

Theorem 3 *Let $V^{\pi_{\text{-}y}}$ be the expected value of policy $\pi_{\text{-}y}$ w.r.t. to the reduced $MDP_{\text{-}y}$ and let V^{π} be the expected value of policy π w.r.t. to the original MDP. Then $V^{\pi_{\text{-}y}}(s) = V^{\pi}(s)$ $\forall s \in S \setminus \{y\}$.*

PROOF To prove the result we expand the recursive definition of expected value, at each step isolating any reference to the redundant state y. We adopt $P_{\pi}(s'|s) = P(s'|s, \pi(s))$ for concision and begin with the definition of expected value for policy π for state $s_0 \in S \setminus \{y\}$:

$$V^{\pi}(s_0) = r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S} P_{\pi}(s_1|s_0) V^{\pi}(s_1)$$

Next we isolate the redundant state y from the sum over future states. Given start state s_0

the expected value of policy π is as follows:

$$\begin{aligned}
V^{\pi}(s_0) &= r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S} P_{\pi}(s_1|s_0) V^{\pi}(s_1) \\
&= r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S \setminus y} P_{\pi}(s_1|s_0) V^{\pi}(s_1) + \gamma P_{\pi}(y|s_0) V^{\pi}(y) \\
&= r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S \setminus y} P_{\pi}(s_1|s_0) V^{\pi}(s_1) \\
&\qquad + \gamma P_{\pi}(y|s_0) \Big[r(y, \pi(y)) + \gamma \sum_{s_2 \in S} P_{\pi}(s_2|y) V^{\pi}(s_2) \Big] \\
&= r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S \setminus y} P_{\pi}(s_1|s_0) V^{\pi}(s') \\
&\qquad + \underbrace{\gamma P_{\pi}(y|s_0) \Big[\gamma \sum_{s_2 \in S} P_{\pi}(s_2|y) V^{\pi}(s_2) \Big]}_{*} \qquad \text{[since } r(y, \pi(y)) = 0\text{]}
\end{aligned}$$

We have removed the redundant state from the original sum over future states at the next step t, however, the value of transitioning to the redundant state (denoted (*)) involves another sum over all future states.

We continue to isolate reference to the redundant state by performing another expansion.

$$\begin{aligned}
V^{\pi}(s_0) &= r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S \setminus y} P_{\pi}(s_1|s_0) V^{\pi}(s_1) \\
&\qquad + \gamma P_{\pi}(y|s_0) \Big[\gamma \sum_{s_2 \in S} P_{\pi}(s_2|y) V^{\pi}(s_2) \Big] \\
&= r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S \setminus y} P_{\pi}(s_1|s_0) V^{\pi}(s_1) \\
&\qquad + \gamma P_{\pi}(y|s_0) \left[\gamma \sum_{s_2 \in S \setminus y} P_{\pi}(s_2|y) V^{\pi}(s_2) + \gamma P_{\pi}(y|y) \Big[\gamma \sum_{s_3 \in S} P_{\pi}(s_3|y) V^{\pi}(s_3) \Big] \right]
\end{aligned}$$

(below we expand and collect terms)

$$\begin{aligned}
&= r(s_0, \pi(s_0)) + \gamma \sum_{s_1' \in S \setminus y} P_{\pi}(s_1'|s_0) V^{\pi}(s_1) \\
&\qquad + \gamma P_{\pi}(y|s_0) \gamma \sum_{s_2 \in S \setminus y} P_{\pi}(s_2|y) V^{\pi}(s_2) \\
&\qquad + \gamma P_{\pi}(y|s_2) \gamma P_{\pi}(y|y) \gamma \sum_{s_3 \in S} P_{\pi}(s_3|y) V^{\pi}(s_3) \\
&= r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S \setminus y} \left[P_{\pi}(s_1|s_0) + P_{\pi}(y|s_0) \gamma P_{\pi}(s_1|y) \right] V^{\pi}(s_1) \\
&\qquad + \gamma P_{\pi}(y|s_0) \gamma P_{\pi}(y|y) \gamma \sum_{s_2 \in S} P_{\pi}(s_2|y) V^{\pi}(s_2)
\end{aligned}$$

We repeat the procedure

$$V^\pi(s_0) = r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S \setminus y} \Bigg[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y) \Bigg] V^\pi(s_1)$$

$$+ \gamma P_\pi(y|s_0)\gamma P_\pi(y|y)\gamma \Bigg[\sum_{s_2 \in S \setminus y} P_\pi(s_2|y)V^\pi(s_2) + P_\pi(y|y)\Big[\gamma \sum_{s_3 \in S} P_\pi(s_3|y)V^\pi(s_3)\Big]\Bigg]$$

(we again expand and collect terms)

$$= r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S \setminus y} \Bigg[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y) \Bigg] V^\pi(s_1)$$

$$+ \gamma P_\pi(y|s_0)\gamma P_\pi(y|y)\gamma \sum_{s_2 \in S \setminus y} P_\pi(s_2|y)V^\pi(s_2)$$

$$+ \gamma P_\pi(y|s_0)\gamma P_\pi(y|y)\gamma P_\pi(y|y)\gamma \sum_{s_3 \in S} P_\pi(s_3|y)V^\pi(s_3)$$

$$= r(s_0, \pi(s_0))$$

$$+ \gamma \sum_{s_1 \in S \setminus y} \Bigg[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y) + P_\pi(y|s_0)\gamma P_\pi(y|y)\gamma P_\pi(s_1|y) \Bigg] V^\pi(s_1)$$

$$+ \gamma P_\pi(y|s_0)\gamma P_\pi(y|y)\gamma P_\pi(y|y)\gamma \sum_{s_2 \in S} P_\pi(s_2|y)V^\pi(s_2)$$

$$= r(s_0, \pi(s_0))$$

$$+ \gamma \sum_{s_1 \in S \setminus y} \Bigg[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y) \sum_{t=0}^{2} \gamma^t P_\pi(y|y)^t \Bigg] V^\pi(s_1)$$

$$+ \gamma P_\pi(y|s_0)\gamma P_\pi(y|y)\gamma P_\pi(y|y)\gamma \sum_{s_2 \in S} P_\pi(s_2|y)V^\pi(s_2)$$

Repeating the procedure once more produces the following:

$$V^\pi(s_0) = r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S \setminus y} \Bigg[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y) \sum_{t=0}^{3} \gamma^t P_\pi(y|y)^t \Bigg] V^\pi(s_1)$$

$$+ \gamma P_\pi(y|s_0)\gamma P_\pi(y|y)\gamma P_\pi(y|y)\gamma P_\pi(y|y)\gamma \sum_{s_2 \in S} P_\pi(s_2|y)V^\pi(s_2)$$

We hypothesize the k^{th} expansion as follows:

$$V^\pi(s_0) = r(s_0, \pi(s_0)) + \gamma \sum_{s_1 \in S \setminus y} \Bigg[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y) \sum_{t=0}^{k} \gamma^t P_\pi(y|y)^t \Bigg] V^\pi(s_1)$$

$$+ \gamma P_\pi(y|s_0)\Big[\gamma P_\pi(y|y)\Big]^k \gamma \sum_{s_2 \in S} P_\pi(s_2|y)V^\pi(s_2),$$

We take an inductive step constructing the $(k+1)^{th}$ expansion in terms of the k^{th} expansion:

$$\begin{aligned}
V^{\pi}(s_0) &= r(s_0,\pi(s_0)) + \gamma \sum_{s_1\in S\setminus y}\left[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y)\sum_{t=0}^{k}\gamma^t P_\pi(y|y)^t\right]V^\pi(s_1) \\
&\quad + \gamma P_\pi(y|s_0)\Big[\gamma P_\pi(y|y)\Big]^k \gamma \sum_{s_2\in S} P_\pi(s_2|y)V^\pi(s_2), \\
&= r(s_0,\pi(s_0)) + \gamma \sum_{s_1\in S\setminus y}\left[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y)\sum_{t=0}^{k}\gamma^t P_\pi(y|y)^t\right]V^\pi(s_1) \\
&\quad + \gamma P_\pi(y|s_0)\Big[\gamma P_\pi(y|y)\Big]^k \gamma \left[\sum_{s_2\in S\setminus y} P_\pi(s_2|y)V^\pi(s_2) + P_\pi(y|y)\Big[\gamma\sum_{s_3\in S}P_\pi(s_3|y)V^\pi(s_3)\Big]\right]
\end{aligned}$$

(we expand and collect terms)

$$\begin{aligned}
&= r(s_0,\pi(s_0)) + \gamma \sum_{s_1\in S\setminus y}\left[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y)\sum_{t=0}^{k}\gamma^t P_\pi(y|y)^t\right]V^\pi(s_1) \\
&\quad + \gamma \sum_{s_2\in S\setminus y} P_\pi(y|s_0)\gamma P_\pi(s_2|y)\Big[\gamma P_\pi(y|y)\Big]^k V^\pi(s_2) \\
&\quad + P_\pi(y|s_0)\Big[\gamma P_\pi(y|y)\Big]^{k+1}\gamma\sum_{s_3\in S}P_\pi(s_3|y)V^\pi(s_3) \\
&= r(s_0,\pi(s_0)) + \gamma \sum_{s_1\in S\setminus y}\left[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y)\sum_{t=0}^{k+1}\gamma^t P_\pi(y|y)^t\right]V^\pi(s_1) \\
&\quad + \gamma P_\pi(y|s_0)\Big[\gamma P_\pi(y|y)\Big]^{k+1}\gamma\sum_{s_2\in S}P_\pi(s_2|y)V^\pi(s_2),
\end{aligned}$$

Given the base cases $k=2$, $k=3$ above, and our inductive step, we have shown that our expression for the k^{th} expansion holds. Finally we take the limit $k\to\infty$:

$$\begin{aligned}
V^{\pi}(s_0) &= r(s_0,\pi(s_0)) + \gamma \sum_{s_1\in S\setminus y}\left[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y)\sum_{t=0}^{\infty}\gamma^t P_\pi(y|y)^t\right]V^\pi(s_1) \\
&\quad + \gamma P_\pi(y|s_0)\cancelto{0}{\Big[\gamma P_\pi(y|y)\Big]^{\infty}}\gamma\sum_{s_2\in S}P_\pi(s_2|y)V^\pi(s_2) \\
&= r(s_0,\pi(s_0)) + \gamma \sum_{s_1\in S\setminus y}\left[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y)\sum_{t=0}^{\infty}\gamma^t P_\pi(y|y)^t\right]V^\pi(s_1) \\
&= r(s_0,\pi(s_0)) + \gamma \sum_{s_1\in S\setminus y}\left[P_\pi(s_1|s_0) + P_\pi(y|s_0)\gamma P_\pi(s_1|y)\frac{1}{1-\gamma P_\pi(y|y)}\right]V^\pi(s_1) \\
&= r(s,\pi(s)) + \gamma\sum_{s'\in S'}P'_\pi(s'|s)V^\pi(s') \\
&= V^{\pi\text{-}y}(s_0)
\end{aligned}$$

Thus we have shown that we can safely reduce an MDP removing a redundant state without altering the value function. $\square$

Appendix B

Random MDP Generation Details

Algorithm 8 gives the details for randomly generating the MDP instances used for empirical analysis. For each state-action-pair, $\lceil \log |S| \rceil$ reachable states are drawn uniformly and a Gaussian is used to generate transition probabilities. The true reward is drawn uniformly from a fixed interval and uncertainty w.r.t. the true reward is created by constraining the reward for each (s, a)-pair independently with bounds drawn randomly. The set of feasible rewards forms a hyper-rectangle.

Algorithm 8: Random Generation of Semi-Sparse Flat-State-Space MDP

```
Input:
$S \leftarrow$ State space
$A \leftarrow$ Action space
$\gamma \leftarrow$ Discount factor
// Specify Transitions
foreach $s \in S, a \in A$ do
    $S' \leftarrow$ Draw $\lceil \log |S| \rceil$ from $S$ without replacement
    foreach $s' \in S'$ do
        $P(s'|s,a) \leftarrow max(0, min(1, N(1/3, 1/5)))$
    end
    // Normalize
    foreach $s' \in S'$ do
        $P(s'|s,a) \leftarrow P(s'|s,a) / \sum_{s' \in S'} P(s'|s,a)$
    end
end
// Specify Rewards
foreach $s \in S, a \in A$ do
    $r(s,a) \leftarrow$ Uniform(0,1)
1   $interval\_size \leftarrow N(1/2, 1/5)$
2   $interval\_above \leftarrow U(0, interval\_size)$
3   $r^{\top}(s,a) \leftarrow min(1, r(s,a) + interval\_above)$
4   $r^{\perp}(s,a) \leftarrow max(0, r^{\top}(s,a) - interval\_size)$
end
$\mathcal{R} \leftarrow \{ r \mid r^{\perp} \leq r \leq r^{\top} \}$
Return: $\langle S, A, P, \mathcal{R}, \gamma, \beta \rangle$
```

Appendix C

RLT Formulations

In this section we describe our application of the reformulation-linearization technique (RLT) (Sherali and Alameddine, 1992) to approximate the max regret computation for IRMDPs with flat (unstructured) reward functions. We then describe how the approach may be extended to compute max regret for IRMDPs with structured reward.

C.1 Max regret for flat IRMDPs

Given a policy $\mathbf{f}$, we wish to linearize the following bilinear program for computing max regret using RLT:

$$\begin{aligned} \underset{\mathbf{g},\mathbf{r}}{\text{maximize}} \quad & \mathbf{g}\cdot\mathbf{r} - \mathbf{f}\cdot\mathbf{r} \\ \text{subject to:} \quad & \gamma\mathbf{E}^{\top}\mathbf{g} + \boldsymbol{\beta} = \mathbf{0},\ \mathbf{g} \geq \mathbf{0} \\ & \mathbf{Cr} \leq \mathbf{d} \end{aligned} \tag{C.1}$$

Reformulation The first phase of the RLT technique constructs valid quadratic constraints by using pairwise products of inequality constraints or products of equality constraints with variables. We begin by re-expressing the linear program (C.1) so as to explicitly enumerate all constraints:

$$\begin{aligned}
&\underset{\mathbf{r},\mathbf{g}}{\text{maximize}} && \mathbf{g}\cdot\mathbf{r} - \mathbf{f}\mathbf{r} && && \text{(C.2a)}\\
&\text{subject to:} && \sum_{s_0} g(s_0) - \gamma P(s_0|\cdot)\mathbf{g} = \beta(s_0) && \forall\ s_0 \in S && \text{(C.2b)}\\
& && \mathbf{c}_i\cdot\mathbf{r} \le d_i && \forall\ i \in \text{rows of } \mathbf{C} && \text{(C.2c)}\\
& && g(s,a) \ge 0 && \forall\ s \in S, a \in A && \text{(C.2d)}
\end{aligned}$$

$P(s_0|\cdot)$ is an $|S||A|$ length vector with an entries sa mapping to $P(s_0|s,a)$; we understand $\mathbf{c}_i$ to be row i of coefficient matrix $\mathbf{C}$ and d_i to be entry i of the coefficient vector $\mathbf{d}$. Let Ω denote the set of constraints (C.2b)–(C.2d). We re-express constraints (C.2c) and (C.2d) as follows:

$$\begin{aligned}
g(s,a) &\ge 0 && \forall\ s \in S, a \in A && \text{(C.3)}\\
d_i - \mathbf{c}_i\cdot\mathbf{r} &\ge 0 && \forall\ i \in \text{rows of } \mathbf{C}
\end{aligned}$$

The constraints in (C.3) are multiplied:

$$g(s,a)\Big[d_i - \mathbf{c}_i\cdot\mathbf{r}\Big] \ge 0 \qquad \forall\ s \in S, a \in A, i \in \text{rows of } \mathbf{C} \qquad \text{(C.4a)}$$

Next the equality constraints are multiplied with each variable $f(s,a)$ and $r(s,a)$.

$$\begin{aligned}
g(s',a')\left[\sum_{s_0} g(s_0) - \gamma P(s_0|\cdot)\mathbf{g} = \beta(s_0)\right] &= 0 && \forall\ s' \in S, a' \in A, s_0 \in S && \text{(C.4b)}\\
r(s',a')\left[\sum_{s_0} g(s_0) - \gamma P(s_0|\cdot)\mathbf{g} = \beta(s_0)\right] &= 0 && \forall\ s' \in S, a' \in A, s_0 \in S && \text{(C.4c)}
\end{aligned}$$

Linearization Next a variable substitution strategy is applied that transforms the generated set of nonlinear constraints (C.4) into a set of linear constraints. We substitute:

$$X^{sa}_{s'a'} = r(s,a)g(s',a') \qquad \forall\, s \in S, a \in A, s' \in S, a' \in A \tag{C.5a}$$

$$Y^{sa}_{s'a'} = r(s,a)r(s',a') \qquad \forall\, s \in S, a \in A, s' \in S, a' \in A \tag{C.5b}$$

$$Z^{sa}_{s'a'} = g(s,a)g(s',a') \qquad \forall\, s \in S, a \in A, s' \in S, a' \in A \tag{C.5c}$$

This linearizes the reformulated problem into the form:

$$\begin{aligned} \underset{\mathbf{r},\mathbf{g},X,Y,Z}{\text{maximize}} \quad & \sum_{s,a} X^{sa}_{sa} - g(s,a)r(s,a) \\ \text{subject to:} \quad & (\mathbf{r},\mathbf{g},X,Y,Z) \in \Omega \cap \Omega_L \end{aligned} \tag{C.6}$$

Where Ω is our original set of constraints and Ω_L is the linearized set of constraints (C.4) under the transformation (C.5). The resulting LP (C.6) is a relaxation of (C.1) in the following sense: given a feasible solution $(\bar{\mathbf{g}}, \bar{\mathbf{r}})$ to the latter problem, there exists $\bar{\mathbf{X}}, \bar{\mathbf{Y}}, \bar{\mathbf{Z}}$ constructed via the substitution (C.5), such that $(\bar{\mathbf{g}}, \bar{\mathbf{r}}, \bar{\mathbf{X}}, \bar{\mathbf{Y}}, \bar{\mathbf{Z}})$ is a feasible solution to the former problem with the same objective. The converse is not necessarily true, thus (C.6) yields an upper bound on (C.1).

C.2 Max regret for IRMDPs with structured reward

We wish to linearize the following cubic program using the reformulation-linearization technique (Sherali and Alameddine, 1992):

$$\begin{aligned} \underset{\mathbf{g},\boldsymbol{\lambda},\boldsymbol{\upsilon}}{\text{maximize}} \quad & \mathbf{g}\cdot\boldsymbol{\lambda}\boldsymbol{\upsilon} - \mathbf{f}\cdot\boldsymbol{\lambda}\boldsymbol{\upsilon} \\ \text{subject to:} \quad & \gamma\mathbf{E}^{\top}\mathbf{g} + \boldsymbol{\beta} = \mathbf{0},\ \mathbf{g} \geq \mathbf{0} \\ & \mathbf{C}_{\lambda}\boldsymbol{\lambda} \leq \mathbf{d}_{\lambda} \\ & \mathbf{C}_{\upsilon}\boldsymbol{\upsilon} \leq \mathbf{d}_{\upsilon} \end{aligned} \tag{C.7}$$

Applying the RLT procedure to a program with a cubic objective (and linear constraints) involves a straightforward extension of the procedure described above in Section C.1. The reformulation step is carried out by taking three-way products of the original linear constraints to form cubic constraints. The linearization step performs variable substitution, creating a new variable for each quadratic and cubic term. The resulting linear program serves as a relaxation that bounds the original objective from above. Further details of each step can be found in Section C.1.

Appendix D

Simulating Website Version Testing

D.1 Constructing a Model of User Behaviour for the Static Website

HTTP requests do not contain enough information about the origin of the request to differentiate between a user typing the URL directly into the browser, and a user clicking on a link in a webpage. However, using the request timestamp we can identify cases where a request likely originated from a user receiving a page from the website and then clicking a link to another page from the website. [1] For example let $s_1, s_2, \dots, s_k$ be such a sequence of requests. It is reasonable to assume that s_{i+1} was initiated by clicking on a link in the response to request s_i. We can then define P_{static} as follows:

$$P_{static}(s'|s) = \frac{\text{count}(s \to s')}{\sum_{s''} \text{count}(s \to s'')}$$

Where count($s_i \to s_j$) is the total instances in which a user requested s_j by clicking on a link in the response to request s_i and count($s_i \to s_\emptyset$) indicates that no further request was by the user

[1]There are some subtleties to identifying followed hyperlinks (vs. direct requests for a URL), however, given the popularity of the dataset there is established prior work on resolving such ambiguities (Kallepalli and Tian, 2001; Xu, Zhang, and Chen, 2010).

during the session. In this static website model the request directly corresponds to the page that is served in response. Thus we can describe P_{static} as the probability that a user will navigate to page s' given that they are on page s.

D.2 Simulating Version Testing

Let P_{visit} be a distribution over the webpages where the probability $P_{visit}(s)$ is proportional to the number of times a page s has been accessed:

$$P_{visit}(s) = \frac{\sum_{s' \in S} \text{count}(s' \rightarrow s)}{\sum_{s' \in S} \sum_{s'' \in S} \text{count}(s' \rightarrow s'')}$$

The set S_{vt} of pages to be version tested is constructed by sampling without replacement from the distribution P_{visit}. Given S_{vt} we wish to simulate the results of testing on each version a of page $s \in S_{vt}$; by varying the contents of the page, a version will alter user behaviour, impacting the distribution $P(s'|s, a)$ over next page page s' to be visited.

To model the impact of each version on user behaviour, we make some assumptions about how (and why) a version is constructed. Intuitively the purpose of a particular version is to boost traffic to some goal page—at the necessary expense of traffic to other pages. Algorithm 9 fleshes out this intuition and details how we randomly select a goal page for each version and simulate the impact on user behaviour.

Algorithm 9: SimulateVersionTest

```
Input:
  $S_g$    The set of goal pages
  $S_{vt}$    The set of pages to version test
  $|A| - 1$    The number of versions per page
  $P(\cdot \mid \cdot\,,\ a_{static})$    The transitions for the underlying (static) webgraph
  $\theta_{imp}$    The mean increase in probability that a version yields toward a goal

foreach page $s \in S_{vt}$ do
    foreach version $a \in A - a_{static}$ do
        $s_g \leftarrow$ uniformly sample goal page from $S_g$ (without replacement for page $s$)
        $\delta \leftarrow \text{Normal}(\theta_{imp}, \frac{\theta_{imp}}{2})$    sample the improvement that version yields
                                    (average version improves traffic by $\theta_{imp}$).
        $\Pr(s_g \mid s, a) \leftarrow \Pr(s_g \mid s, a_{static}) + \delta$
        Let $s_1, \ldots, s_\ell$ be the pages linked to by $s$ (excluding $s_g$)
        $\delta_1, \ldots, \delta_\ell \leftarrow$ randomly partition $\delta$ probability mass such that $\delta = \delta_1 + \cdots + \delta_\ell$
        for $i = 1, \ldots, \ell$ do
            $P(s_i \mid s, a) \leftarrow \Pr(s_i \mid s, a_{static}) - \delta_i$
        end
    end
end
Return: $P$
```

Appendix E

MDP Specifications

E.1 Assistive Technology

Section 5.5.1 discusses a simplified model of the COACH system (Boger et al., 2005), whose general goal is to guide a patient with dementia through a task with ℓ steps, such as hand-washing, using verbal or visual cues, while minimizing intrusion. Prompts can be issued at increasing levels of intrusiveness until (at the highest level k) a caregiver is called to assist the person in task completion. The assessment conducted in Section 5.5.1 uses the settings $\ell = 10$ and $k = 4$. This results in action space with prompt levels $A = \{0, 1, 2, 3\}$. The state is defined by three variables $\mathcal{S} = \langle T, D, F \rangle$; $T \equiv \{0, 1, \ldots, 9\}$ is the number of tasks steps successfully completed by the person, $D \equiv \{0, 1, 2, 3, 4, 5+\}$ is the *delay* (time taken during the current step), and $F \equiv \{0, 1, 2, 3\}$ tracks whether a prompt at a specific level was attempted on the current task step, but failed to immediately get the person to the next step. We use an infinite horizon Markov decision process with a discount of $\gamma = 0.95$ and a deterministic starting state of $\mathrm{s}_0 = \langle t = 0, d = 0, f = 0 \rangle$.

The dynamics of our model express the following intuitions. Each action will cause a *progress* transition to the next step (setting delay and failed-prompt to zero), or a *stall* transition (same step with delay increased by one). The probability of reaching the next step with action

$a = n$ is higher than $a = n - 1$ since more intrusive prompts have a better chance of facilitating progress; however, progress probability decreases as delay increases. Reaching the next step after prompting is less likely if a prompt has already failed at the current step. We define an *initial probability* $\theta = 0.8$ of progress on each step. On step t, given a delay of d, a failure setting of f, and a prompt level of $a \in \{0, k-1\}$, we define the probability of *progress* to the next step as:

$$\begin{aligned} P(t'=t+1, d'=0, f'=0 \mid t, d, f, a) = \quad & \theta \quad \times && \big[\text{intial progress probability}\big] \\ & \Big((a < f)\ ?\ (0.75) : 1\Big) \quad \times && \big[\text{impact of failed prompt}\big] \\ & (0.9)^d \quad + && \big[\text{impact of delay}\big] \\ & a\frac{1-\theta}{k} && \big[\text{impact of prompt}\big] \end{aligned}$$

Where the ternary operator $a\ ?\ b : c$ selects b when a is true and c otherwise. Alternately, the probability of a *stall* is defined as:

$$P(t'=t, d'=1, f'=a \mid t, d, f, a) = 1 - P(t'=t+1, d'=0, f'=0 \mid t, d, f, a)$$

For state $\langle t, d, f \rangle$ and action a, either a *progress* or *stall* transition occurs; the probability of any other transition given state $\langle t, d, f \rangle$ and action a is zero. A discount of $\gamma = 0.95$ is used. A special progress transition is defined for states where step $t = \ell$ that transitions to a special absorbing end state $s_{\otimes}$ (where self transition has probability one) that has zero reward—effectively ending the task. The action $a = k$ represents calling a care-giver to intervene in the task; the selection of this action in any state will transition the system to the end state $s_{\otimes}$.

d	$\upsilon_d(d)$
0	0.00
1	-0.15
2	-0.30
3	-0.45
4	-0.60
5+	-0.75

Table E.1: Local Delay Penalty $\upsilon_d(d)$

a	$\upsilon_p(a)$
0	0.00
1	-0.05
2	-0.10
3	-0.75

Table E.2: Local Prompt Penalty $\upsilon_p(a)$

The additive reward function is defined as follows:

$$r(t, d, f, a) = r_g(t) + r_d(d) + r_p(a),$$

where: $r_g(t)$ is a positive "task completion" reward (with a value of 1.0 if $t = \ell$ task, zero otherwise); $r_d(d)$ is a negative "delay" penalty; and $r_p(a)$ is a negative "prompting" penalty associated with prompting the person. Typically, $r_p(a = k)$ is a very large negative cost for calling the caregiver (relative to other costs); and the sub-reward functions $r_d(d)$ and $r_p(a)$ are both assumed to be monotonic (in delay and prompting level, respectively). Each sub-reward function is the product of a scaling constant and local utility function: $r_i = \lambda_i \upsilon_i$. Attribute F does not occur in the reward function, so requires no elicitation. The tables E.1 and E.2 specify the local delay penalty function and local prompt penalty function. The respective calibration constants are $\lambda_d = 0.2$ and $\lambda_p = 1.0$.

To simulate an IRMDP, the uncertain reward function $\mathcal{R}$ is created by generating upper and lower bounds for each reward parameter independently. These bounds are generated using the labelled steps 1–4 from Algorithm 8 for randomly generating IRMDPs. These steps essentially: 1) generate of an uncertain interval whose size is normally distributed and 2) uniformly place the interval around the true reward point .

$P(d_1'\|d_1)$	High	Med	Low
High	0.70	0.25	0.05
Med	**0.45**	0.10	**0.45**
Low	0.05	**0.25**	**0.70**

Table E.3: Markov Chain for Server 1

$P(d_2'\|d_2)$	High	Med	Low
High	0.70	0.25	0.05
Med	**0.60**	0.10	**0.30**
Low	0.05	**0.45**	**0.50**

Table E.4: Markov Chain for Server 2

E.2 Autonomic Computing

Section 5.5.2 discusses a simplified autonomic computing task (Kephart and Chess, 2003) that involves allocating computing or storage resources to servers as computing demands from clients change over time.

The assessment conducted in Section 5.5.2 uses two *application server elements* e_1, e_2, and three *units of resource* which may be assigned to the server elements (plus a "zero resource"). An allocation is specified by $\mathbf{n} = \langle n_1, n_2 \rangle$ where $n_i \in \{0, 1, 2, 3\}$ and $n_1 + n_2 \leq 3$. Finally there are three *demand levels* at which each server element can operate. A full specification of demand levels is denoted $\mathbf{d} = \langle d_1, d_2 \rangle$ where $d_i \in \{1, 2, 3\}$.

A state is given by the current allocation of resources and current demand levels for each server: $\mathbf{x} = \langle n_1, n_2, d_1, d_2 \rangle$. Actions are allocations $\mathbf{m} = \langle m_1, m_2 \rangle$ of up to 3 units of resource to the 2 application servers, such that $m_i \in \{0, 1, 2, 3\}$ and $m_1 + m_2 \leq 3$. Uncertainty in demand is exogenous and the action in the current state uniquely determines the allocation in the next state. Thus the transition function is composed of i Markov chains $\Pr(d_i' \mid d_i)$, one for the demand at each server element: $P(n_1', n_2', d_1', d_2') = P(d_1'|d_1)P(d_2'|d_2)$.

The tables E.3 and E.4 specify the Markov chain for each application server element. Entry (i, j) in the table represents $P(j|i)$. We refer to demand level 1 as *low*, level 2 as *med* and level 3 as *high*. The Markov chain for application server element 2 expresses increased likelihood to move to high demand from the medium demand state (and to medium demand from the low demand state); these differences are highlighted in tables in bold.

The reward $r(\mathbf{n}, \mathbf{d}, \mathbf{m}) = u(\mathbf{n}, \mathbf{d}) - c(\mathbf{n}, \mathbf{d}, \mathbf{m})$ is composed of a positive utility $u(\mathbf{n}, \mathbf{d})$ and the negative cost $c(\mathbf{n}, \mathbf{d}, \mathbf{m})$. The cost $c(\mathbf{n}, \mathbf{d}, \mathbf{m})$ is the sum of the costs of taking away one unit

$v_1(n_1, d_1)$	High	Med	Low
0	0.00	0.00	0.00
1	1.00	0.50	0.25
2	1.50	1.00	0.50
3	1.75	1.50	1.00

Table E.5: Local Utility for Server 1

$v_2(n_2, d_2)$	High	Med	Low
0	0.00	0.00	0.00
1	1.25	0.63	0.31
2	1.88	1.25	0.63
3	2.19	1.88	1.25

Table E.6: Local Utility for Server 2

of resource from each server element at each time step: $c(\mathbf{n}, \mathbf{d}, \mathbf{m}) = \sum_i \max(0, n_i - m_i)$. We assume that the cost term is known. The utility term $u(\mathbf{n}, \mathbf{d})$ can be factored into local utility functions $v_i(n_i, d_i)$ for each server i. In this setting, utility functions are defined with respect to a common unit (potential revenue), so there is no need for calibration: $\boldsymbol{\lambda} = \mathbf{1}$. Thus, $u(\mathbf{n}, \mathbf{d}) = v_1(n_1, d_1) + v_2(n_2, d_2)$.

The tables E.5 and E.6 specify the local utility function for each application server element. The general trends to observe are that: 1) there is more local utility realized when higher demand is met with a high allocation of resource, and 2) application server element 2 produces slightly more local utility than application server element 1 given the same demand and allocation profile.

To simulate an IRMDP we generate the uncertain reward function $\mathcal{R}$ as follows. The interval establishing the upper and lower bounds on each uncertain reward parameter is generated by applying steps 1–4 from Algorithm 8 for randomly generating IRMDPs. These steps essentially: 1) generate of an uncertain interval whose size is normally distributed and 2) uniformly place the interval around the true reward point .

Bibliography

David Andre and Stuart Russell. Programmable reinforcement learning agents. In *Proceedings of the Fifteenth Annual Conference on Neural Information Processing Systems (NIPS-01)*, pages 1019—1025, 2001. 31

ACM SIGCOMM. Internet Traffic Archive. `http://ita.ee.lbl.gov/index.html`. 148

David Avis. lrs: A revised implementation of the reverse search vertex enumeration algorithm. In *Polytopes–Combinatorics and Computation*, pages 177–198. Birkhauser-Verlag, 2000. 127

Andrew Bagnell, Andrew Ng, and Jeff Schneider. Solving uncertain Markov decision problems. Technical Report CMU-RI-TR-01-25, Carnegie Mellon University, Pittsburgh, 2003. 35, 83

Ralphen Becker, Shlomo Zilberstein, Victor R. Lesser, and Claudia V. Goldman. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, 22:423–455, 2004. 139

Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957. 26

J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962. 48, 84

Jennifer Boger, Pascal Poupart, Jesse Hoey, Craig Boutilier, Geoff Fernie, and Alex Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1293–1299, Edinburgh, 2005. 41, 113, 115, 190

Jennifer Boger, Pascal Poupart, Jesse Hoey, Craig Boutilier, Geoff Fernie, and Alex Mihailidis. A planning system based on Markov decision processes to guide people with dementia through activities of daily living. *IEEE Transactions on Information Technology in Biomedicine*, 10(2):323–333, 2006. 41, 129

Craig Boutilier. A POMDP formulation of preference elicitation problems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 239–246, Edmonton, 2002. 19, 22

Craig Boutilier and Richard Dearden. Using abstractions for decision-theoretic planning with time constraints. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 1016–1022, Seattle, 1994. 31

Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1104–1111, Montreal, 1995. 31

Craig Boutilier, Ronen I. Brafman, and Christopher Geib. Prioritized goal decomposition of Markov decision processes: Toward a synthesis of classical and decision theoretic planning. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1156–1162, Nagoya, 1997. 31

Craig Boutilier, Thomas Dean, and Steve Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999. 30, 31, 67, 98, 99

Craig Boutilier, Rajarshi Das, Jeffrey O. Kephart, Gerald Tesauro, and William E. Walsh. Cooperative negotiation in autonomic systems using incremental utility elicitation. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 89–97, Acapulco, 2003a. 2, 41

Craig Boutilier, Richard S. Zemel, and Benjamin Marlin. Active collaborative filtering. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 98–106, Acapulco, 2003b. 23

Craig Boutilier, Ronen Brafman, Carmel Domshlak, Holger Hoos, and David Poole. Cp-networks: A tool for representing and reasoning with conditional *Ceteris Paribus* preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004a. 14

Craig Boutilier, Tuomas Sandholm, and Rob Shields. Eliciting bid taker non-price preferences in (combinatorial) auctions. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, pages 204–211, San Jose, CA, 2004b. 47

Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. Regret-based utility elicitation in constraint-based decision problems. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 929–934, Edinburgh, 2005. 21, 94, 118

Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artifical Intelligence*, 170(8–9):686–713, 2006. 2, 4, 5, 47, 81, 82, 94, 100, 107

Craig Boutilier, Kevin Regan, and Paolo Viappiani. Online feature elicitation in interactive optimization. In *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML-09)*, pages 73–80, Montreal, 2009a. 168

Craig Boutilier, Kevin Regan, and Paolo Viappiani. Preference elicitation with subjective features. In *Proceedings of the 3rd ACM Conference on Recommender Systems (RecSys09)*, pages 341–344, New York, 2009b. 168

Craig Boutilier, Kevin Regan, and Paolo Viappiani. Simultaneous elicitation of preference features and utility. In *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 1160–1167, Atlanta, 2010. 94, 168

Ronen Brafman and Moshe Tennenholtz. R-max- a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2003. 33

Darius Braziunas and Craig Boutilier. Minimax regret-based elicitation of generalized additive utilities. In *Proceedings of the Twenty-third Conference on Uncertainty in Artificial Intelligence (UAI-07)*, pages 25–32, Vancouver, 2007. 100, 118, 120

Darius Braziunas and Craig Boutilier. Elicitation of factored utilities. *AI Magazine*, 29(4): 79–92, 2008. 118

Darius Braziunas and Craig Boutilier. Assessing regret-based preference elicitation with the UTPREF recommendation system. In *Proceedings of the Eleventh ACM Conference on Electronic Commerce (EC'10)*, pages 219–228, Cambridge, MA, 2010. 94, 118, 166

Donald E. Brown and Chelsea C. White. An expert system approach to boiler design. *IEEE Transactions on Systems, Man and Cybernetics,*, 17(2):293–297, 1987. 22

Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 1023–1028, Seattle, 1994. 38

Urszula Chajewska. *Acting Rationally with Incomplete Utility Information*. PhD thesis, Stanford University, Stanford, 2002. 19

Urszula Chajewska and Daphne Koller. Utilities as random variables: Density estimation and structure discovery. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 63–71, Stanford, 2000. 19, 21

Urszula Chajewska, Daphne Koller, and Ronald Parr. Making rational decisions using adaptive utility elicitation. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, pages 363–369, Austin, TX, 2000. 2

Hsien-Te Cheng. *Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, Vancouver, 1988. 59, 71, 125, 127, 139

Vašek Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1983. 127

Adam Coates, Pieter Abbeel, and Andrew Ng. Learning for control from multiple demonstrations. 2008. 40, 94

Thomas Dean and Robert Givan. Model minimization in Markov decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 106–111, Providence, 1997. 31

Erick Delage and Shie Mannor. Percentile optimization in uncertain Markov decision processes with application to efficient exploration. In *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML-07)*, pages 225–232, Corvallis, OR, 2007. 19, 37, 73, 81

Mukund Deshpande and George Karypis. Selective Markov models for predicting web page accesses. *ACM Transactions on Internet Technology (TOIT)*, 4(2):163–184, 2004. 157

Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000. 31

Chelsea C. White Edward A. Sykes. Specifications of a knowledge system for packet-switched data network topological design. In *Proceedings of Expert Systems Government Symposium*, pages 102–110, 1985. 22

Boi Faltings, Pu Pearl, and Marc Torrens. Designing example-critiquing interaction. In *Proceedings of the Nineth International Conference on Intelligent User Interfaces*, pages 22–29, 2004. 24

Peter C. Fishburn. Interdependence and additivity in multivariate, unidimensional expected utility theory. *International Economic Review*, 8:335–342, 1967. 99, 100, 120

Peter C. Fishburn. *Utility Theory for Decision Making*. Wiley, New York, 1970. 11, 13, 14, 15

Marshall L. Fisher. Interactive optimization. *Annals of Operations Research*, 5(1):541–556, 1986. 77

Simon French. *Decision Theory*. Halsted Press, New York, 1986. 11, 13, 15, 16, 81

Christophe Gonzales and Patrice Perny. GAI networks for utility elicitation. In *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR2004)*, pages 224–234, Whistler, BC, 2004. 118

Paul E. Green and Vithala R. Rao. Conjoint measurement for quantifying judgmental data. *Journal of Marketing Research*, 8(3):355–363, 1971. 22

Paul E. Green and V. Srinivasan. Conjoint analysis in consumer research: Issues and outlook. *Journal of Consumer Research*, 5(2):103–123, 1978. 23

Paul E. Green and V. Srinivasan. Conjoint analysis in marketing: New developments with implications for research and practice. *Journal of Marketing*, 54(4):3–19, 1990. 23

Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored Mdps. In *Journal of Artificial Intelligence Research*, volume 19, pages 399–468, 2003a. 76, 120

Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19(10):399 – 468, 2003b. 32

Vijaykumar Gullapalli and Andrew Barto. Convergence of indirect adaptive asynchronous value iteration algorithms. In *Proceedings of the Eighth Annual Conference on Neural Information Processing Systems (NIPS-94)*, page 695, 1994. 27

Ian Hacking. Jacques bernoulli's art of conjecturing. *The British Journal for the Philosophy of Science*, 22(3):209–229, 1971. 18

Ahmed Hassan, Rosie Jones, and Kristina Lisa Klinkner. Beyond DCG: User behavior as a predictor of a successful search. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 221–230. ACM, 2010. 157

Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 220–229, Madison, WI, 1998. 152

Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 279–288, Stockholm, 1999. 30, 31

Vera Hollink, Maarten van Someren, and Bob J. Wielinga. Navigation behavior models for link structure optimization. *User Modeling and user-adapted Interaction*, 17(4):339–377, 2007. 149, 158

Hillary A. Holloway and Chelsea C. White, III. Question selection for multiattribute decision-aiding. *European Journal of Operational Research*, 148:525–543, 2003. 22

Laura M. Holson. Putting a bolder face on Google. New York Times, February 28 2009. 141

Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960. 24, 25, 27

Garud N. Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 30 (2):1–21, 2005. 34, 83

Vijay S. Iyengar, Jon Lee, and Murray Campbell. Q-Eval: Evaluating multiple attribute items using queries. In *Proceedings of the Third ACM Conference on Electronic Commerce*, pages 144–153, Tampa, FL, 2001. 20

Phillipe Jorion. *Value at Risk: The New Benchmark for Controlling Market Risk*. Irwin Chicago, 1997. 19

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998. 32, 59, 64, 65, 139

Chaitanya Kallepalli and Jeff Tian. Measuring and modeling usage and reliability for statistical web testing. *IEEE Transactions on Software Engineering*, 27(11):1023–1036, 2001. 187

Henry Kautz, Bart Selman, and Mehul Shah. Referral web: Combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63–65, 1997. 23

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2):209–232, 2002. 33

Ralph L. Keeney and Howard Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York, 1976. 2, 11, 13, 14, 99, 100, 160

Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36 (1):41–52, 2003. 41, 116, 193

Omar Zia Khan, Pascal Poupart, and James P. Black. Minimal sufficient explanations for factored Markov decision processes. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling*, 2009. 79

Omar Zia Khan, Pascal Poupart, and James P. Black. Automatically generated explanations for Markov decision processes. In Enrique Sucar, Eduardo Morales, and Jesse Hoey, editors, *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, chapter 7, pages 144–163. IGI Global, 2011. 79

Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997. 23

Brendan McMahan, Geoffrey Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pages 536–543, Washington, DC, 2003. 5, 36, 73, 83, 84

Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Solving very large weakly coupled Markov decision processes. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 165–172, Madison, WI, 1998. 31

George L. Nemhauser and Laurence A. Wolsey. *Integer Programming and Combinatorial Optimization*. Wiley, New York, 1988. 48

Andrew Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-00)*, pages 663–670, Stanford, CA, 2000. 4, 32, 39, 40, 73, 94, 134, 168

Vahab S. Mirrokni Nikolay Archak and S. Muthukrishnan. Budget optimization for online advertising campaigns with carryover effects. In *The Eleventh ACM SIGECOM International Conference on Electronic Commerce, Harvard*, 2010. 157

Arnab Nilim and Laurent El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(1):780–798, 2005. 36, 83

Eunsoo Oh and Kee-Eung Kim. A geometric traversal algorithm for reward-uncertain Mdps. In *Proceedings of the Twenty-seventh Conference on Uncertainty in Artificial Intelligence (UAI-11)*, 2011a. 134, 135, 137

Eunsoo Oh and Kee-Eung Kim. A geometric traversal algorithm for reward-uncertain Mdps. In *Proceedings of the Twenty-seventh Conference on Uncertainty in Artificial Intelligence (UAI-11)*, Barcelona, 2011b. 75

Ronald Parr. Flexible decomposition algorithms for weakly coupled Markov decision processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 422–430, Madison, WI, 1998. 152

Relu Patrascu, Craig Boutilier, Rajarshi Das, Jeffrey O. Kephart, Gerald Tesauro, and William E. Walsh. New approaches to optimization and utility elicitation in autonomic computing. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 140–145, Pittsburgh, 2005. 94

M. Perkowitz and O. Etzioni. Adaptive sites: Automatically learning from user access patterns. In *Proc. 6th Int. World Wide Web Conf., Santa Clara, California*, 1997. 149, 158

Marek Petrik and Shlomo Zilberstein. A bilinear programming approach for multiagent planning. *Journal of Artificial Intelligence Research*, 35(1):235–274, 2009. 139

Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, volume 18, pages 1025–1032, 2003. 139

Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, 27(1):335–380, 2006. 139

Pascal Poupart. Approximate value-directed belief state monitoring for partially observable Markov decision processes. Master's thesis, University of British Columbia, Vancouver, 2000. 39

Pascal Poupart, Craig Boutilier, Relu Patrascu, and Dale Schuurmans. Piecewise linear value function approximation for factored MDPs. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 292–299, Edmonton, 2002. 32, 76, 120

Pearl Pu, Boi Faltings, and Marc Torrens. User-involved preference elicitation. In *IJCAI-03 Workshop on Configuration*, Acapulco, 2003. 23

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994. 24, 27, 28, 29, 64, 66, 171

Martin L. Puterman and M.C. Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24:1127–1137, 1978. 27

Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2586 – 2591, 2007. 40, 74, 94

Nathan Ratliff, Andrew Bagnell, and Martin Zinkevich. Maximum margin planning. pages 729 – 736, 2006. 40, 73, 94

Kevin Regan. Myoptically optimal policy comparison queries uing setwise max regret. University of Toronto Tech Report, 2011. 97

Kevin Regan and Craig Boutilier. Regret-based reward elicitation for Markov decision processes. In *Proceedings of the Twenty-Second Conference on Neural Information Processing Systems - Workshop on Model Uncertainty and Risk in Reinforcement Learning*, 2008. 43, 76, 165

Kevin Regan and Craig Boutilier. Regret-based reward elicitation for Markov decision processes. In *Proceedings of the Twenty-fifth Conference on Uncertainty in Artificial Intelligence (UAI-09)*, pages 454–451, Montreal, 2009. 43, 74, 76, 97, 165

Kevin Regan and Craig Boutilier. Robust policy computation in reward-uncertain MDPs using nondominated policies. In *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 1127–1133, Atlanta, 2010. 5, 43, 74, 76, 120, 165

Kevin Regan and Craig Boutilier. Eliciting additive reward functions for markov decision processes. In *Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence (IJCAI-11)*, 2011a. 76, 120, 165

Kevin Regan and Craig Boutilier. Robust online optimization of reward-uncertain MDPs. In *Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence (IJCAI-11)*, 2011b. 140, 165

James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. Dynamic critiquing. In *Proceedings of the European Conference on Case-Based Reasoning*, pages 763 – 777, 2004. 24

James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. Incremental critiquing. *Knowledge-Based Systems*, 18(4-5):143–151, 2005. 24

James Reilly, Kevin McCarthy, and Barry Smyth. Evaluating compound critiquing recommenders: a real-user study. In *Proceedings of the Eighth ACM Conference on Electronic Commerce (EC'07)*, pages 114–123, 2007. 24

Jason Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the Twenty-second International Conference on Machine Learning (ICML-05)*, 2005. 23

Maya Rupert, Amjad Rattrout, and Salima Hassas. The web from a complex adaptive systems perspective. *Journal of Computer and System Sciences*, 74(2):133–145, 2008. 149, 158

Ahti Salo and Raimo P. Hämäläinen. Preference programming through approximate ratio comparisons. *European Journal of Operational Research*, 82:458–475, 1995. 2, 4

Paul A. Samuelson. Consumption theory in terms of revealed preference. *Economica*, 15(60): 243–253, 1948. 4

Ramesh R. Sarukkai. Link prediction and path analysis using Markov chains. *Computer Networks*, 33(1):377–386, 2000. 157

Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, pages 285–295, 2001. 23

Leonard J. Savage. The theory of statistical decision. *Journal of the American Statistical Association*, 46(253):55–67, 1951. 18

Leonard J. Savage. *The Foundations of Statistics*. Wiley, New York, 1954. 3

Dale Schuurmans and Relu Patrascu. Direct value-approximation for factored MDPs. In *Proceedings of the Fifteenth Annual Conference on Neural Information Processing Systems (NIPS-01)*, volume 14, 2001. 32

Hanif D. Sherali and Amine Alameddine. A new reformulation-linearization technique for bilinear programming problems. *Journal of Global Optimization*, 2:379–410, 1992. 52, 54, 183, 186

Satinder P. Singh and David Cohn. How to dynamically merge Markov decision processes. In *Advances in Neural Information Processing Systems 10*, pages 1057–1063. MIT Press, Cambridge, 1998. 31

Paul Slavic, Baruch Fischhaff, and Sarah Lichtenstein. Behavioral decision theory. *Annual Review of Psychology*, 28(1):1–39, 1977. 2

Mathis T.J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24(1):195–220, 2005. 139

Richard Sutton, Doina Precup, and Satinder Singh. Intra-option learning about temporally abstract actions. pages 556–564, 1998. 152

Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988. 33

Gerald Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134:181 – 199, 2002. 32

Aimo Torn and Antanas Zilinskas. *Global Optimization*. Springer, Berlin, 1989. 74

Olivier Toubia, John Hauser, and Duncan Simester. Polyhedral methods for adaptive choice-based conjoint analysis. (4285-03), 2003a. 82

Olivier Toubia, Duncan I. Simester, John R. Hauser, and Ely Dahan. Fast polyhedral adaptive conjoint estimation. *Marketing Science*, 22(3):273–303, 2003b. 20

Olivier Toubia, John R. Hauser, and Duncan I. Simester. Polyhedral methods for adaptive choice-based conjoint analysis. *Journal of Marketing Research*, 41(1):116–131, 2004. 20

Paolo Viappiani and Craig Boutilier. Regret-based optimal recommendation sets in conversational recommender systems. In *Proceedings of the 3rd ACM Conference on Recommender Systems (RecSys09)*, pages 101–108, New York, 2009. 87, 88, 89, 94, 95, 172

Paolo Viappiani, Boi Faltings, and Pearl Pu. Preference-based search using example-critiquing with suggestions. *Journal of Artificial Intelligence Research*, 27:465–503, 2006a. 24

Paolo Viappiani, Boi Faltings, and Pearl Pu. Evaluating preference-based search tools: a tale of two approaches. In *Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI-06)*, pages 205–211, 2006b. 24

John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, 1944. 11, 12, 13

Abrahaml Wald. *Statistical Decision Functions*. Wiley, New York, 1950. 17, 18

Tianhan Wang and Craig Boutilier. Incremental utility elicitation with the minimax regret decision criterion. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 309–316, Acapulco, 2003. 2, 167

Youwei Wang, Dingwei Wang, and W. H. Ip. Optimal design of link structure for e-supermarket website. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 36(2):338–355, 2006. 149, 158

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992. 33

Huan Xu and Shie Mannor. Parametric regret in uncertain Markov decision processes. In *48th IEEE Conference on Decision and Control*, pages 3606–3613, Shanghai, 2009. 4, 5, 45, 59, 60, 62, 74, 161

Lei Xu, Weifeng Zhang, and Lianjie Chen. Modeling users' visiting behaviors for web load testing by continuous time markov chain. In *Web Information Systems and Applications Conference (WISA), 2010 7th*, pages 59–64. IEEE, 2010. 187

W Zhang and T Dietterich. High-performance job-shop scheduling with a timedelay td (lambda) network. In *Proceedings of the Tenth Annual Conference on Neural Information Processing Systems (NIPS-96)*, 1996. 32

B. Zhou, J. Chen, J. Shi, H. Zhang, and Q. Wu. Website link structure evaluation and improvement based on user visiting patterns. In *Proceedings of the 12th ACM conference on hypertext and hypermedia*, pages 241–244. ACM, 2001. 149, 158

Brian Ziebart, Andrew Maas, Andrew Bagnell, and Anind Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the Twenty-third AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 1433 – 1438, 2008. 40, 74, 94