The Optimal Reward Problem: Designing Effective Reward for Bounded Agents

by

Jonathan Daniel Sorg

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2011

Doctoral Committee:

Professor Satinder Singh Baveja, Co-Chair
Professor Richard L. Lewis, Co-Chair
Professor John E. Laird
Professor Michael P. Wellman
Associate Professor Thad A. Polk

Acknowledgments

This dissertation could not have happened without the insight or guidance of my advisors, Satinder and Rick. The lessons I learned from them will be invaluable in the next stage of my life. Thanks also to my dissertation committee members John Laird, Thad Polk, and especially Michael Wellman for their insightful comments.

I'd like to thank my lab-mates, Erik Talvitie, Britton Wolfe, David Wingate, Matt Rudary, and Vishal Soni for their mentorship, and Jeshua Bratman and Rob Cohn for their partnership. I wouldn't have learned nearly as much without our engaging discussions. Thanks also to David Karmol, Mike Maxim, and Miller Tinkerhess for helping with the Spaces framework, in which most of the empirical work was done. Special thanks to Akram Helou who coded the Bait-or-Fish experiment and helped with the optimization algorithm used in Chapter 3.

Thanks to my family, Mom, Dad, Matt, and Brandon for supporting me thorough all these years, and to Grandma for sending me cake. Thanks to all of my friends, but especially to my roommates Eric and Wen, with whom I shared many hours studying at the living room table.

And most of all thank you Kristen, for achieving this with me.

Table of Contents

List of Tables

Table

List of Figures

Figure

List of Algorithms

Abstract

In the field of reinforcement learning, agent designers build agents which seek to maximize reward. In standard practice, one reward function serves two purposes. It is used to evaluate the agent and is used to directly guide agent behavior in the agent's learning algorithm.

This dissertation makes four main contributions to the theory and practice of reward function design. The first is a demonstration that if an agent is bounded—if it is limited in its ability to maximize expected reward—the designer may benefit by considering two reward functions. A designer reward function is used to evaluate the agent, while a separate agent reward function is used to guide agent behavior. The designer can then solve the Optimal Reward Problem (ORP): choose the agent reward function which leads to the greatest expected reward for the designer.

The second contribution is the demonstration through examples that good reward functions are chosen by assessing an agent's limitations and how they interact with the environment. An agent which maintains knowledge of the environment in the form of a Bayesian posterior distribution, but lacks adequate planning resources, can be given a reward proportional to the variance of the posterior, resulting in provably efficient exploration. An agent with poor modeling assumptions can be punished for visiting the areas of the state space it has trouble modeling, resulting in better performance.

The third contribution is the Policy Gradient for Reward Design (PGRD) algorithm, a convergent gradient ascent algorithm for learning good reward functions. Experiments in multiple environments demonstrate that using PGRD for reward optimization yields better agents than using the designer's reward directly as the agent's reward. It also outperforms the use of an evaluation function at the leaf-states of the planning tree.

Finally, this dissertation shows that the ORP differs from the popular work on potential-based reward shaping. Shaping rewards are constrained by properties of the environment and the designer's reward function, but they generally are defined irrespective of properties of the agent. The best shaping reward functions are suboptimal for some agents and environments.

Chapter 1

Introduction

The field of Artificial Intelligence (AI) is largely dedicated to the study of designing artificial autonomous agents which can act on behalf of the agent designer. The agent designer's goals implicitly define a preference ordering over possible interactions between agent and its environment: given two different sequences of interactions, the agent designer is able to decide which (if any) of the two is preferred. The agent designer's task, therefore, is to design the best agent with respect to these preferences.

In order to design such an agent, the designer must consider how the agent's behavior interacts with the environment. Due to the complexity of this interaction, it is rare that the agent designer knows the most preferred or optimal behavior in a form that allows him or her to directly program it into the agent. It may be complicated to compute the optimal behavior given the designer's preferences, or the designer may lack knowledge about the environment. To accommodate these difficulties, it is often more robust to build goals and purposes into the agent itself, rather than specifying agent behavior directly. This allows the agent designer to instruct the agent what to do without having to tell the agent how to do it. The agent can use its goals for computing good behavior, a process referred to as *planning*, and can also use its goals to update its behavior based on its experience, a process referred to as *learning*.

The field of AI has developed formalized representations for specifying agent goals and has designed algorithms for finding good behaviors given these goals. Although the principles in this dissertation broadly apply to many forms of goal representations, I focus here on a method of goal representation common to the fields of Reinforcement Learning (RL), operations research, and decision-theoretic planning. Specifically, I capture an agent's goals as the maximization of cumulative expected *reward*. The reward can be captured concisely using a *reward function*, which I define formally in the next chapter.

I have chosen this representation of agent goals for two reasons. One, as evidenced by its broad use, the reward function is a powerful representation of goals. The

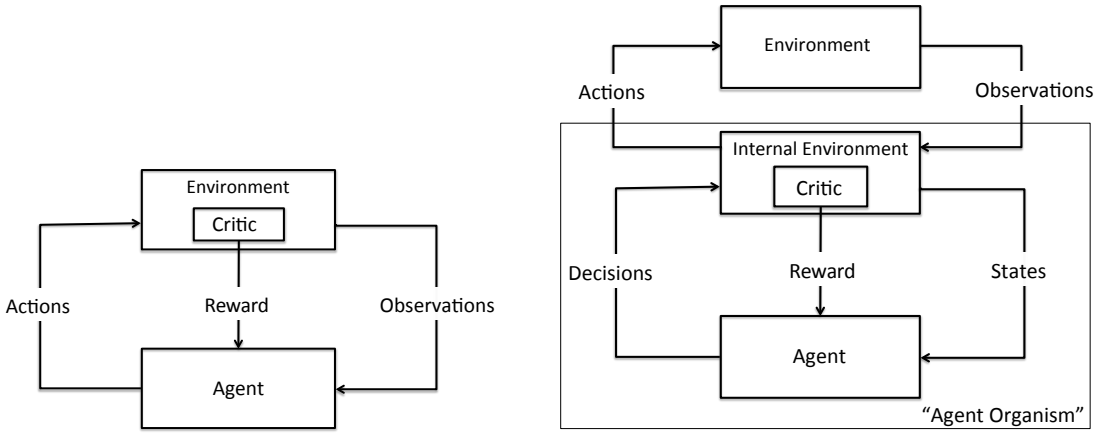


Figure 1.1: Diagrams depicting agent-environment interactions; adapted from Barto et al. (2004). Left panel: The reward is supplied to the agent from its environment. Right panel: A refinement in which the environment is factored into internal and external environments, with all reward coming from the former.

reward hypothesis states, "That all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward)" (Sutton, 2004). Two, the mathematical results and planning/learning algorithms from the RL community provide an excellent foundation for many of the results in this dissertation.

1.1 Designing Reward Functions

In the standard view of a reinforcement learning problem, the reward is an *external* signal, one that originates from the environment, or from some other source, such as an external critic. This concept is illustrated in the left panel of Figure 1.1. The agent interacts with its environment by taking actions and receiving observations. The critic observes the interaction and sends the agent reward proportional to how well it is doing. The agent's goal is to maximize the reward received from the external critic. As I discuss below, the reward is arguably better viewed as being internal to the agent.

All Reward is Internal to the Agent. Sutton and Barto (1998) point out that an RL agent should not be considered to be an entire animal or robot. Instead, an RL agent is the learning component of the brain. The remaining physical components are

are separate from the learning component, but are still a part of the broader "agent organism." This concept is illustrated in the right panel of Figure 1.1. In this view, the environment is factored into two components: the internal environment and the external environment. The agent organism is comprised of the internal environment and the RL agent. The agent organism takes actions in the physical world by applying torque to its arms, for example, while the RL agent itself makes decisions, such as the decision to move in a particular direction. Similarly, the internal environment is responsible for physically sensing the environment and converting those sensations into signals that the RL agent can interpret. In an agent design scenario, the designer is responsible for building the entire agent organism, including the internal environment.

It is rare that an external critic exists in the environment to send the appropriate signal. In practice, designers build an autonomous critic. Furthermore, the interpretation of any external signal as a reward signal is part of agent design. Thus the critic is built along with the other components of the agent, and it is better considered to be a part of the agent organism. Throughout this dissertation, I adopt the notion that an agent designer is tasked with designing an agent organism, reward function included, but I often drop the word "organsim" and refer to the entire agent organism as an "agent."

Choosing the Reward Function. The process of converting a designer's preferences into a reward function is not trivial. First, a designer has flexibility in how to encode his or her preferences as a reward function—many reward functions can encode the same preference ordering. However, depending on the agent, reward functions that theoretically map to the same preference ordering may lead to different behavior. Second, a designer can frequently benefit by assigning preferences to the artificial agent that differ from his or her own.

Consider the example of a designer building a chess-playing agent with the objective of building an agent that wins the game as often as possible. As one solution, the designer could build a critic which rewards the agent for winning the game and punishes the agent for losing. This feedback is very sparse. Thus, it can be expensive for an agent's planning algorithm to account for, and it it can be difficult for learning to utilize. It might help to also reward the agent for taking the opposing player's queen, providing more dense feedback, even though queen capture is not the designer's goal. This can be viewed as specifying a subgoal.

Once an agent designer takes this first step into designing an agent which has goals that differ from his or her own, it opens a world of possibilities. Suppose the designer

is building a learning agent. It might help to explicitly reward the agent for *learning*. This type of reward forms the basis of *intrinsic motivation* (Ryan and Deci, 2000; Barto et al., 2004) and *artificial curiosity* (Schmidhuber, 1991c). Suppose the agent has difficulty modeling certain situations, leading the agent to repeatedly make poor decisions in these situations. A designer can punish the agent for visiting situations it has trouble modeling, a motivation similar to anxiety, causing it to avoid these situations. This dissertation explores reward functions with these characteristics.

Preference Elicitation. Finally, a difficulty in reward function design that arises in practice occurs because a designer often does not fully understand his or her own preferences. Consider the example of designing an autonomous car-driving agent. I should probably punish the agent for crashing the car and reward the agent for reaching its destination. A designer may produce this as a first attempt, but should the agent also be punished for speeding? For driving on the shoulder? For switching lanes too often? Often, an agent designer will build an agent that achieves its defined goals, only to discover that its goals were defined poorly. The agent will drive on the shoulder, for example, because it has not been told not to.

The challenge of efficiently gathering an agent designer's preferences is known as the preference elicitation problem. It has been studied recently in the context of RL and relates to inverse reinforcement learning (Ng and Russell, 2000; Abbeel and Ng, 2004; Ramachandran and Amir, 2007; Regan and Boutilier, 2009). However, the preference elicitation problem is not the focus of this dissertation. Here, I assume that the designer understands his or her own preferences and wants to build the best agent possible given these preferences.

1.2 Summary of this Dissertation

The Optimal Reward Problem. Agent design implicitly involves the reward design process, though designers do not always acknowledge this explicitly. Existing work on this topic lacks a formal framework for evaluating and comparing particular reward function choices, because it lacks the formal separation between agent and designer preferences. In Chapter 2, I present an optimization problem, the *Optimal Reward Problem* (ORP), which provides a criterion for selecting reward functions when designing artificial agents. The ORP unifies these prior approaches, in the sense that these approaches can all be seen as solutions of constrained instances of the ORP.

The optimal reward problem is the starting point for this dissertation, from which a number of questions arise, including:

- When can a designer stand to benefit from solving the optimal reward problem?
- How does one solve the optimal reward problem?
- How do reward design approaches compare to alternative approaches for improving agents?

The remainder of the dissertation, described below, addresses these questions while proposing solutions to specific instances of the optimal reward problem and proposing novel reward design algorithms.

Overcoming Agent Constraints. In common practice, agent designers frequently choose a simple, direct encoding of the agent designer's preferences as the agent's reward function, rather than choosing an optimal reward function. Diverting from this common practice has a cost in terms of an increased complexity of the design process; therefore, it is important to understand the magnitude and the nature of the benefit of diverting from this common practice.

I show that the benefit of reward design depends on the computational constraints placed on the agent. If an agent is bounded—if it has constraints that prevent it from achieving good performance using the standard definition of reward—then the designer stands to benefit from solving the optimal reward problem.

Critically, understanding the agent's limitations helps in designing a reward function for overcoming these limitations. Sutton (1990), for example, realized this when he designed a non-stationary reward function for an agent that otherwise assumed stationarity to encourage persistent exploration in a time-changing environment. I expand on this notion, showing that many different types of agent limitations can be overcome through reward design and give examples of rewards to mitigate these limitations. Chapter 3 presents a framework for characterizing limits on agents and how such limits can be overcome through reward function design. The framework is related to the notion of bounded optimality (Russell and Subramanian, 1995). Next, Chapter 3 presents agents with various limitations being improved through reward function design. The reward functions are designed to take into account the interaction between the agent's limitations and the environment it is placed in.

Deriving Reward Functions. Although the reward functions in Chapter 3 are developed heuristically, in Chapter 4 I show how to design a reward function using

this notion of agent limitations that has strong theoretical guarantees. The reward is derived by taking an optimal, but computationally infeasible agent design (Bayes-optimal learning), developing a computationally simpler approximation of that agent, and then using a reward to help overcome the damage done by the approximation. The reward explicitly encourages exploration (and learning) by rewarding the agent for visiting areas of the environment which have high uncertainty in its model. The resulting agent achieves efficient exploration (in a PAC sense) while taking into account its prior knowledge of the world.

An Online Reward Design Algorithm. In Chapter 5, I present an algorithm for solving the optimal reward problem online during an agent's lifetime. The algorithm has per-step computational complexity proportional to the agent's own computational cost and doesn't require additional off-line knowledge. Furthermore, the algorithm is capable of improving reward functions in agents with a variety of limitations.

Scaling Up and Comparing to Similar Approaches. In Chapter 6 I extend the gradient method developed in Chapter 5 to two large-scale planning agent architectures, including UCT (Kocsis and Szepesvári, 2006), a state-of-the-art architecture used in a number of applications (Gelly and Silver, 2008; Finnsson and Björnsson, 2008). These algorithms are used to compare the reward design approach to another common approach to overcoming computational constraints on agents—that of applying an evaluation function to the leaf node of a planning tree. I show that the reward design approach subsumes the leaf evaluation function approach, and I show that for a variety of agent architectures, including UCT, the reward design approach performs better than the leaf-evaluation function approach. I demonstrate this in multiple environments, including Othello, a two-player game with approximately 10^{28} states. The results in Chapter 6 demonstrate that the popular class of reward functions known as potential-based shaping reward functions, from the work of Ng et al. (1999), does not always contain an optimal reward function.

Conclusion. Finally, I conclude in Chapter 7. In addition to a summary of the results and conclusions from the dissertation, this chapter contains a comparison of the optimal reward problem to the broader bounded agent design problem. I conclude with open questions and future directions.

Chapter 2

The Optimal Reward Problem

In the standard RL definition, there is one reward function which serves two purposes. First, it defines the preferences in the designer—it is used to evaluate an agent's performance. Second, it is used to directly guide behavior of the agent—changing the reward function changes the agent's behavior. The use of one reward function *confounds* these two purposes of the reward function. By construction, the standard RL definition assumes that the agent's goals and the designer's goals should be the same.

This dissertation shows that the goals of the agent need not be the same as the designer's. Although I assume the designer's goals are given, the designer is free to choose the agent's goals. The designer's objective is to design the best-performing agent possible. Thus, a designer faces the Optimal Reward Problem (ORP)—the optimization problem of choosing the best reward function with respect to the agent designer's goals.

The optimal reward problem forms the foundation upon which the remainder of this dissertation is built. This chapter lays that foundation. In section 2.1, I formally define the ORP. I discuss properties of optimal reward functions and demonstrate these properties by approximately solving an example optimal reward problem in section 2.2. In section 2.3, I discuss how the ORP framework can be used to analyze and unify existing reward design attempts while discussing related work.

The optimal reward problem was originally presented by Singh et al. (2009) in the context of a natural agent being selected through evolution. In this dissertation, I present it in the context of a single agent design scenario. Whereas Singh et al. (2009) captured environment uncertainty by assuming it was selected from a distribution of environments, here I assume the designer is designing an agent for one environment. The two formulations are equivalent, because a distribution of environments can be modeled as a single environment with a coin toss at the beginning of time that determines which of the multiple environments the agent is in.

2.1 Optimal Reward Problem Definition

An agent design scenario consists of the environment, the agent, and the agent designer. I formally introduce the relevant aspects of each of these before formally presenting the ORP itself.

Environment and Agent At each time step, an agent G receives an observation $o \in \mathcal{O}$ from its environment M, takes an action $a \in \mathcal{A}$, and repeats this process until a (possibly infinite) time horizon. In general, the state of an environment at time step k, denoted h_k, is the history of interaction $o_1 a_1 \cdots o_{k-1} a_{k-1} o_k$ to time step k. The set of possible environment states is the set of all finite sequences of interaction. Although representations of state more compact than full histories are used in practice, for this discussion it is convenient to use full history as state for complete generality. I denote the set of all finite sequences $\mathcal{H} = \{(\mathcal{O} \times \mathcal{A})^i : i \in \mathbb{N}\}$ and the set of all sequences, including infinite-length sequences $\mathcal{H}_\infty = \{(\mathcal{O} \times \mathcal{A})^i : i \in \mathbb{N} \bigcup \infty\}$.

The transition dynamics of M define a probability distribution over possible next states h_{k+1} as a function of h_k and a_k. Specifically, let D_X denote the set of distributions over set X. Let T be the transition dynamics of the environment. Specifically, T defines a distribution over the next observation given the current state and action, $T : \mathcal{H} \times \mathcal{A} \to D_{\mathcal{O}}$, and the next state is defined by appending the previous action and sampled observation to the current state.

An agent G is a mapping from histories to distributions over actions. Specifically, $G : \mathcal{H} \to D_{\mathcal{A}}$. When an agent is placed in an environment, it samples actions according to this definition given the current history, and the environment responds with an observation sampled according to the environment dynamics. This interaction occurs repeatedly, producing a distribution over histories. I use the notation $M\langle G \rangle$ to refer to the history generated when agent G acts in environment M. Thus, $M\langle G \rangle \in \mathcal{H}_\infty$ is a random variable. I use the notation $h \sim M\langle G \rangle$ to denote that h is a sample history resulting from this interaction.

The Agent Designer's Goals. Axiomatic decision theory shows that (under mild conditions) a designer's preference ordering over agent behavior traces can be expressed as a utility function that maps behavior traces to scalar values, such that a behavior trace with a larger associated utility value is preferred to a behavior trace with a lower associated utility value (Mas-Colell et al., 1995). Without loss of generality, then, the preferences of an agent designer can be represented with such utility functions.

In the field of RL, the agent designer's utility function over histories is decomposed into two functions. One function, the *objective reward function* R_O, assigns immediate reward as a function of its current history, $R : \mathcal{H} \to \mathbb{R}$. The other function, the accumulation function U, defines how these rewards are accumulated over time. Together, the objective reward function and the accumulation function combine to make up the designer's utility function, which I refer to as the *objective return function* $U_{R_O} : \mathcal{H}_\infty \to \mathbb{R}$. For example, in the average reward case, the objective return for an infinite-length history h is $U_{R_O}(h) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} R_O(h_i)$. And the discounted return function, for example, is $U_{R_O}(h) = \lim_{N\to\infty} \sum_{i=1}^{N} \gamma^{i-1} R_O(h_i)$, for some discount factor $\gamma \in [0, 1)$. These definitions are easily extended to finite-length trajectories.

Thus, the utility of a agent designer receives from an agent is equal to the objective return of the trajectory produced when the agent is placed in the target environment. The goal of the designer is to maximize its expected utility, or expected objective return: $\mathbb{E}[U_{R_O}(h)|h \sim M\langle G\rangle]$.

This decomposition of utility into reward and accumulation functions is redundant: there are multiple choices of reward-function and accumulation-function that lead to the same return function U_{R_O}. One reason designers decompose the utility function in this way is so that in the traditional RL setting, the reward may be incrementally sent to the agent as feedback.

This formulation assumes that the designer's utility is definable in terms of features of the agent's behavior trace that the agent can observe. For the purposes of this dissertation, this assumption helps to compare the agent's reward function (defined below) with the designer's utility function. This assumption is not a requirement of the optimal reward problem; a designer can in principle have a utility function defined in terms of features of the environment that the agent cannot observe.

The Agent's Goals. The main departure of this work from the conventional framing of the agent design problem is that the agent's goals are defined independently from the designer's. Just as in the case of the designer, the agent's utility function is decomposed into a reward function R and an accumulation function U, which together make up the agent's return function U_R.

The objective of an agent is to maximize its expected utility, or expected return: $\mathbb{E}[U_R(h)|h \sim M\langle G\rangle]$. Much of the progress in the fields of reinforcement learning and operations research has been devoted towards improving agents at optimizing objectives of this form. I refer to such an agent as an RL agent. Although this term is usually restricted to agents that *learn*, in this work, an RL agent is any agent that

attempts to maximize its expected return, regardless of whether it learns.

In the typical framing of reinforcement learning, the agent's reward function is a fixed, defined property of the problem—the agent is assigned the objective reward. Here, the agent's reward function is an important parameter of agent design. I denote that the reward function is an agent parameter using the notation $G(R)$.

The designer selects the reward function from some set of reward functions $\mathcal{R}$. Although this could in general be from the set of all reward functions, defined by all mappings from finite histories to real numbers, this set is generally too large to search in practice. Thus, I consider searching over constrained sets of rewards in this dissertation[1]. Each instance of an ORP is over a potentially different, constrained search set of reward functions $\mathcal{R}$, to be chosen by the agent designer. One of the goals in later chapters of this dissertation is to analyze in specific instances how the choice of $\mathcal{R}$ interacts with the agent architecture.

Although the accumulation function is also a parameter that affects the agent's goals, I do not focus on optimizing this parameter in this work. One reason for this is that, often, the accumulation function is restricted as a part of the agent architecture[2].

The Optimal Reward Problem. The optimal reward problem arises because the choice of the agent's reward function affects the agent's action choices, and this in turn affects the distribution over environment histories. The designer's objective return function is defined over histories, thus, the designer's choice of reward function affects the return obtained by the designer. The designer's objective is to choose the reward function that produces the best expected objective return.

The optimal reward problem is expressed succinctly with the following equation:

$$R^* = \arg\max_{R \in \mathcal{R}} \mathbb{E}\left[U_{R_O}(h) | h \sim M\langle G(R)\rangle\right]. \tag{2.1}$$

In words, an optimal reward function R^* is one that maximizes the designer's expected objective return for some environment M, agent G, objective return function U_{R_O}, and search set of reward functions $\mathcal{R}$.

[1]Correspondingly, the ORP could have been called the constrained optimal reward problem, or the optimal constrained reward problem, but I use the term "optimal reward problem" to refer to all ORPs, both over constrained sets of rewards and unconstrained.

[2]In fact, it is frequently the case that in standard practice, the accumulation function imposed by an agent architecture doesn't match the accumulation function in the designer's objective return function. For example, agent designers frequently use Q-learning agents with discounted return functions and evaluate their agents by accumulating undiscounted reward (Mahadevan, 1994). This is because Q-learning requires the use of the discounted accumulation function function to obtain convergence in many environments (Bertsekas and Tsitsiklis, 1996).

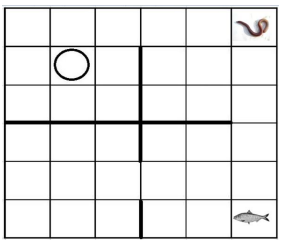


Figure 2.1: Fish-or-Bait Environment. The agent, represented by the circle, can learn to eat bait (easy) or fish (more difficult).

The optimal reward problem requires very little in the form of agent assumptions and comes with a simple guarantee: an optimal reward function performs at least as well as the objective reward function, the conventional choice, as long as the objective reward is in the search set $\mathcal{R}$. This guarantee holds whether or not the accumulation functions of the agent and the designer are the same. The proof is straightforward given the definition of the optimal reward function. It is easy to enforce the constraint that $R_O \in \mathcal{R}$, and in fact all the experiments in this dissertation satisfy this constraint. In the next chapter, I discuss additional constraints on the choice of $\mathcal{R}$.

In practice, designers will not be able to find R^* exactly, and must instead approximate it. This is what is done in the experiments in this dissertation. I denote an approximately optimal reward function $\hat{R}^*$. Chapter 5 introduces an efficient on-line method for approximately solving the optimal reward problem. Prior to that chapter, I resort to inefficient off-line brute-force techniques.

In the next section, I demonstrate additional properties of optimal reward functions through examples.

2.2 Properties of Optimal Reward Functions

In this section, I present a number of salient properties of optimal reward functions by solving the optimal reward problem in a set of experiments. In summary, the experiments in this section demonstrate that:

1. Using an optimal reward results in a better agent than using the objective reward function.

2. Optimal reward functions are sensitive to the environment, the agent, and the designer's objective return.

3. An optimal reward function does not necessarily preserve the preference ordering of the designer's objective return. In other words, an optimal reward can often reflect very different goals than the designer's.

The Environment and the Agent Designer's Goals

Figure 2.1 shows a representation of a 6×6 grid world called the Fish-or-Bait Environment. In the top right corner of the grid is an inexhaustible supply of worms (bait), and at the bottom right corner is an inexhaustible supply of fish. The thick lines represent barriers that have to be navigated around. The agent is shown as a circle. In each location the agent can move deterministically `north`, `south`, `east`, or `west`. Any action that takes the agent off the grid or crosses a barrier fails with no resulting movement. In addition to the movement actions, the agent has actions `pick-up` and `eat`. At the worm location, `pick-up` results in the agent carrying a worm, and `eat` results in the agent eating a worm. The agent cannot carry more than one worm at a time. If the agent executes `eat` at the fish location, it succeeds in eating one fish, but only if it is carrying a worm; otherwise the action fails. At any other location, `eat` results in the agent consuming the worm if it is carrying one. The agent is *satiated* for one time step after eating a fish, *hungry* for one time step after eating a worm, and *famished* at all other times. The agent observes its (x, y) grid-location, whether it is carrying a worm, and its level of hunger. Thus, the agent faces an MDP and its observations can be used to compactly represent the state of the environment. When this is the case, in this dissertation, I use the letter s to denote state.

The objective reward function assigns 0.04 reward each time a worm is eaten (being in state *hungry*), 1.0 reward each time a fish is eaten (being in state *satiated*), and 0 reward when the agent is *famished*. The accumulation function is the sum of reward obtained over some horizon, described below.

Varying the Horizon

One of the goals of this experiment is to demonstrate that the optimal reward is affected by parameters of the environment and by the designer's objective. In Fish-or-Bait world, both can be demonstrated by varying a parameter called the horizon. In the experiment, the agent has a limited amount of time to collect its food. After some fixed horizon, its opportunities for obtaining objective return cease. This choice of horizon could be modeled in two equivalent ways. One could define the objective return as the sum of objective reward over the fixed horizon. Alternatively, one could

define the objective return as a sum over an infinite-length history, and model the horizon as a deterministic state transition at some fixed time t to some absorbing state that produces 0 objective reward. The first method corresponds to a choice of the agent designer's objective return, and the second corresponds to a change in the environment.

Because this horizon choice can be modeled equivalently as a choice of the designer's objective return or as a change in the environment dynamics, these experiments can be used to examine an optimal rewards' dependence on both of these parts of the optimal reward problem. The experiments examine the effect of the choice of horizon by exploring the range of horizons from $1,000$ to $50,000$ in steps of $1,000$.

The Agent

The agent uses look-up-table-based Q-learning (Sutton and Barto, 1998). Q-learning works by maintaining estimates $Q(s, a)$ of the expected return, given that the agent starts in state s, takes action a, and follows an optimal policy thereafter.

Given an observed transition from s to s' given a, the agent observes reward $R(s')$ and updates the Q-function according to the rule: $Q(s, a) \leftarrow \alpha(R(s') + \gamma \max_{a' \in \mathcal{A}} Q(s', a')) + (1 - \alpha)Q(s, a)$, for some discount factor γ and learning rate α. At each step, the agent selects the greedy action $\arg\max_{a \in \mathcal{A}} Q(s, a)$ in the current state s with probability $(1 - \epsilon)$ and a random action with probability ϵ, to help it explore while learning. In the experiments, the agent starts with a Q-function that is uniformly 0. The experiments use $\gamma = 0.99$, $\alpha = 0.1$, and $\epsilon = 0.1$, unless noted otherwise.

The Reward Design Space

The search space of reward functions, $\mathcal{R}$, is defined as the set of all mappings from each of the three mutually disjoint hunger levels to scalars. Without loss of generality, each scalar is restricted to lie between -1.0 and 1.0. At each time step, the agent receives reward equal to the scalar corresponding to its current hunger level. Note that the requirement that the reward design space contain R_O is satisfied by the choice of $\mathcal{R}$.

In this experiment, the optimal reward problem is solved by discretized brute-force search. For each level of hunger, the following values are searched over: the point 0, points in the range $\pm[0.01, 0.1]$ in increments of 0.01, and points in the range $\pm[0.2, 1]$ in increments of 0.1. These intervals and increments were determined through several

iterations of the experiment. For each reward, the agent is placed in the environment and the return is empirically estimated by repeatedly generating sample histories and averaging their returns.

Searching over Initial Q-functions

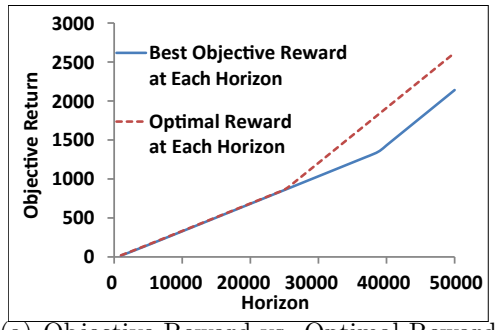
When translating his or her goals into an objective reward function, the designer has flexibility in that some transformations of the objective return function lead to the same preference ordering over histories. One of the objectives of the experiments is to show that an optimal reward function represents different goals than the designer's goals, rather than simply the best translation of designer goals into rewards.

I search over simple translations of the objective reward function, and compare an optimal reward function to the best translation of objective reward. This simple translation is guaranteed to preserve the preference ordering over histories in the designer's objective return function. More precisely, I search over the selection of R_O by fixing the reward for being *satiated* (eating fish), *hungry* (eating worms), and *famished* to to $1.0 + c$, $0.04 + c$, and c, respectively and optimizing the value of $c \in [-1, 1]$ separately for each finite horizon.

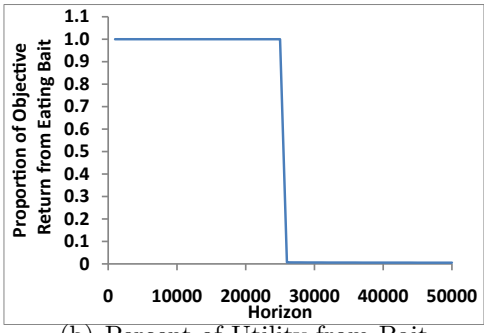
This is not an exhaustive search over all reward functions which result in identical preference orderings over histories, but this set has a strong impact on agent performance. Shifting the rewards by c is equivalent to shifting the value of the discounted sum of rewards by $c/(1-\gamma)$. In Q-learning agents, the agent's initial value function estimate affects the agent's initial exploratory behavior (Sutton and Barto, 1998). The transformation explored effectively allows for optimistic or pessimistic initialization of the value function (which is initialized to 0 in all experiments).

Results

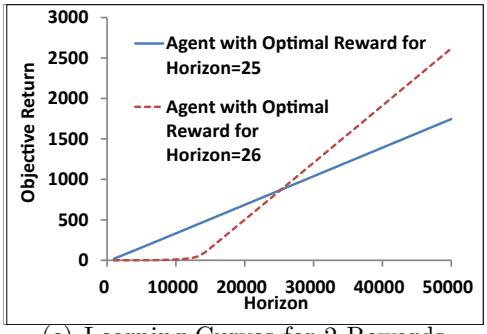
The six panels in Figure 2.2 show the main results. Figure 2.2(a) plots for each horizon value the mean (over 200 runs) objective return for the agent with the best reward function for that horizon and the objective return for the agent with the best objective reward for that horizon. I emphasize that the two curves in Figure 2.2(a) are not standard learning curves. The x-axis is not time steps; each point on the x-axis corresponds to a separate experiment with a distinct horizon. The points are connected for visual clarity.



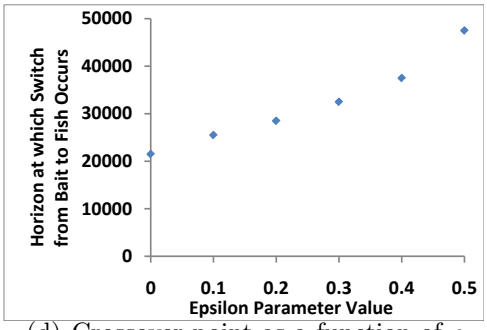
(a) Objective Reward vs. Optimal Reward



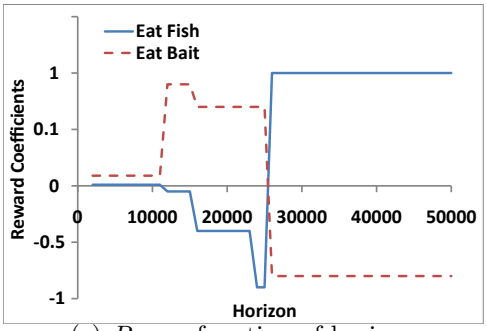
(b) Percent of Utility from Bait



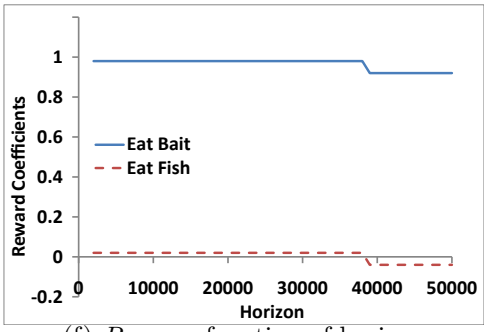
(c) Learning Curves for 2 Rewards



(d) Crossover point as a function of ϵ



(e) R as a function of horizon



(f) R_O as a function of horizon

Figure 2.2: Results for the Fish-or-Bait Domain.

Dominance of optimal reward functions. The optimal reward agent outperforms the best objective reward agent for all values of the horizon. Although the difference between the two is small for the first 25,000 horizon values, the difference is strictly in favor of the optimal reward at all horizons.

Sensitivity to the the environment and the designer's objective. Figure 2.2(a) has another interesting property: the low constant initial advantage is followed by a sharp rise in the advantage at a horizon of 26,000, which is followed by a roughly constant absolute advantage of about 450 after a horizon of 40,000. This results from an interesting intuitive property of the Fish-or-Bait domain. Specifically, the behavior to eat bait is rather simple—find the worm location and then stay there eating. The behavior to eat fish is more complex—find the worm location, pick up a worm and carry it to the fish location without eating the worm at any time, eat the fish, and then go back to the worm location to repeat the cycle. Thus, it is far easier for the agent to learn to eat bait than to learn to fish. However, the longer the horizon, the more it becomes worth it for the agent to learn to fish because there is time within the horizon both to learn to fish and to exploit the greater reward that comes from fishing.

Figure 2.2(b) shows this effect of horizon. It plots, for each optimal reward at each horizon, the proportion of objective return obtained by eating bait. For horizons of 25,000 or less almost all the objective return comes from eating bait. For horizons of 26,000 and above almost all the objective return comes from eating fish.

To illustrate that this is indeed what is happening, Figure 2.2(c) compares the performance (learning curves) of two specific agents. The first uses the approximately optimal reward function $\hat{R}^*$ found for a horizon of 25,000, i.e., just *before* the shift from eating bait to fishing. The second uses the $\hat{R}^*$ found for a horizon of 26,000, i.e., just *after* the shift from eating bait to fishing. As can be seen in Figure 2.2(a), the first agent quickly learns to find the bait location and steadily eat there to achieve a constant objective return increment of 0.04, while the second agent takes some time (about 12,000 steps) to learn how to fish and then achieves a constant rate of objective return by fishing that is higher than that of eating bait. By time 26,000 the second agent's increased rate of objective return helps it overtake the total objective return of the first agent. Another effect of the relative difficulty of learning to fish versus learning to eat bait is seen in Figure 2.2(a): the agent using an optimal reward function can learn to eat fish and gain the higher rate of objective return much sooner (at a horizon of 26,000) than an agent using the objective reward (at a horizon of

40,000).

As stated above, because the horizon can be induced both by choosing the designer's goals appropriately or by modifying the environment, these experiments demonstrate both that an optimal reward depends on the objective return and that it depends on the environment.

Sensitivity to agent parameters. Figure 2.2(d) shows the sensitivity of the approximately optimal reward $\hat{R}^*$ to other parameters of the agent, in this case the exploration rate parameter ϵ. The larger the ϵ, the more often the agent performs a random action. This makes it more difficult for the agent to learn to fish because it has to avoid eating the bait for the many steps it takes to carry the bait to the fish pond. As seen in Figure 2.2(d), the horizon at which the optimal-reward induced crossover from eating bait to eating fish occurs increases as a function of ϵ. As it gets harder to learn to fish, the optimal reward chooses to eat bait at longer horizons.

Optimal Reward Functions Specify a Different Preference Orderings Figure 2.2(e) plots the optimal reward coefficients for eating fish and eating bait as a function of horizon. For the 12,000 steps it takes the agent to learn to first get fish, the optimal reward gives a small positive reward to eating bait and no reward for eating fish because there is no point in rewarding eating-fish up until this horizon.

Something very interesting happens for horizons between 12,000 and 25,000, when it is possible to learn to eat fish but there isn't enough time to exploit that learning. The optimal reward makes it very rewarding to eat bait and very costly to eat fish. This ensures that for this range of horizons the agent eats bait and avoids fish.

From 26,000 steps onwards this polarity reverses and the agent highly rewards eating fish and highly negatively rewards eating bait. The latter negative reward serves to make sure that the agent is not distracted from the goal of learning to fish. Note that in the middle range of horizons the optimal reward in effect reverses the desirability of bait versus fish expressed in the objective return function.

The general point here is that the *optimal reward function need not be a preference-order-preserving transformation of* R_O.

For comparison to Figure 2.2(e), which plots the optimal reward coefficients, Figure 2.2(f) plots as a function of horizon the same two coefficients for the best objective reward, which by definition preserves the relative ordering of fish and bait in the objective return function. The switch from slight positive reward for eating bait to slight negative reward for eating bait at about 40,000 steps is crucial, because

as explained above, prior to that horizon the best objective reward agent should eat bait and after that horizon, it should avoid eating bait. Although this switch will not affect the Q-learning's convergent behavior (which it does not have time to reach), it will affect the agent's early exploration behavior. Given that the agent's Q-function is initialized to 0, this switch will make the agent less likely to explore eating bait when the horizon is greater than $40,000$.

Discussion

The results for the Fish-or-Bait domain illustrate the three consequences of using optimal rewards. First, using an optimal reward function can lead to significantly better agents as measured by the objective return function when compared to agents using the objective reward function. Second, an optimal reward function depends on the details of the objective return function, but the nature of this dependence is not necessarily simple: the best reward function need not preserve the preference ordering of the objective return function. Third, an optimal reward function depends on the parameters of the agent as well as properties of the environment.

2.3 Related Work

Agent designers have long been experimenting with the idea of assigning goals to their agents which differ from their own. Some of these techniques are motivated by traits found in natural agents, such as curiosity (Schmidhuber, 1991c) and other forms of intrinsic motivation (Barto et al., 2004). Some of these techniques provide strong theoretical guarantees about the performance of the resulting agent (Ng et al., 1999; Kearns and Singh, 1998; Brafman and Tennenholtz, 2001). In general, however, prior to work on the optimal reward problem (Singh et al., 2009; Sorg et al., 2010a), these attempts did not explicitly account for the presence of the agent designer and the optimization problem arising from the separation of agent goals from designer goals.

The optimal reward problem provides a concrete framework for analyzing and unifying these prior approaches. In this section, I compare existing work on reward design and goal design more broadly and relate them to each other and to the optimal reward problem. I conclude this section with a short discussion comparing the optimal reward problem in the context of agent design to related ideas arising from the study of natural agents.

I divide related work on reward design in artificial agents into three categories: (1)

reward shaping, (2) intrinsically motivated RL, and (3) PAC (Probably Approximately Correct) methods.

2.3.1 Reward Shaping

One commonly used method for modifying the agent's reward function is known as *reward shaping* (Gullapalli and Barto, 1992; Mataric, 1994; Randløv and Alstrøm, 1998; Ng et al., 1999). The idea of shaping, which is borrowed from behavioral psychology (Skinner, 1938), is to give the learning agent a series of progressively difficult tasks building up to the most difficult task of ultimate interest. Reward shaping, therefore, is the act of giving an agent an easier task through modification of the reward function.

For example, Randløv and Alstrøm (1998) trained an agent to navigate a simulated bicycle to a goal by additionally providing the agent with a small reward bonus for traveling in the correct direction. They note that designing shaping rewards for this purpose is not trivial, because when calibrated poorly, the shaping rewards can lead to undesired behavior. For example, without also punishing the bicycle agent for traveling in the wrong direction, it went in circles rather than reaching the goal.

In a foundational paper on reward shaping, Ng et al. (1999) solved this problem by describing necessary and sufficient conditions on an additive modification to a reward function such that it does not change the corresponding optimal behavior (but it may change the transient behavior of a learning agent). Specifically, an additive modification satisfies these conditions if it is of the potential-based form. I describe the potential-based form in more detail in Chapter 6 Since the publication of Ng et al.'s work, agent designers have used shaping rewards of the potential-based form (Laud, 2004; Konidaris and Barto, 2006; Marthi, 2007; Elfwing et al., 2008; Grześ and Kudenko, 2008, 2009).

Importantly, Wiewiora (2003) showed that for a large class of agents which update estimates of a value function (such as the Q-learning agent, above), for every potential-based reward function there is an equivalent initialization of the value function that results in identical agent behavior. Value function initialization is a common practice in RL agent design with powerful but transient effects on agent behavior. His results suggest (as Ng et al. do) that finding the best shaping reward is equivalent to finding the value function of the optimal policy.

Along these lines, one commonality that reward shaping works tend to share is that the reward functions are based on features of the environment and hence are usually environment-dependent, though some of the above-cited work (Konidaris and Barto,

2006; Marthi, 2007) have moved in the direction of developing shaping rewards in abstracted state spaces. This stands in contrast to the rewards introduced in Chapter 3, which are agent-centric and environment-independent.

Agent Assumptions. The central guarantee of potential-based shaping rewards—that they preserve the optimal policy of the objective reward function—does not make any reference to the type of agent used. However, in many practical situations, agents are not capable of achieving the optimal policy with respect to the reward function they are given due to various constraints. In the Fish-or-Bait experiments, for example, the Q-learning agents did not have enough time to learn to convergence. These are the situations of interest in this dissertation. I discuss the issue of constraints in more detail in the next chapter.

Automatic Potential-Based Reward Optimization. There are many reward functions which satisfy the potential-based criterion. Recent work on reward shaping has addressed the problem of automatically finding good shaping rewards without human input. Konidaris and Barto (2006) developed a system for using a representation that is shared across a sequence of tasks to learn a value function that is shared across environments. This knowledge is then provided to the agent in the form of a potential-based reward function (as in the result from Wiewiora, 2003). Several authors later developed similar methods which work in a single environment by using experience to learn a MDP model (Marthi, 2007; Grześ and Kudenko, 2008). This model is then used to compute a value function, which is supplied to the agent using a potential-based reward function in a similar fashion. Elfwing et al. (2008) select potential-based rewards from a parameterized set using a genetic programming algorithm.

Comparing Potential-Based Reward Shaping to the Optimal Reward Problem. Shaping rewards have generally been developed for the purpose of accelerating learning in a particular environment. The form of rewards used has been explicitly developed with the intention of not changing the long-term behavior of an agent which is capable of learning the optimal policy. In contrast, I show that when an agent is limited, it is sometimes beneficial to modify the long-term behavior of an agent. In the next chapter, I demonstrate that optimal rewards can be used to improve the performance of agents with limitations in addition to the inability to learn efficiently.

Reward functions of the potential-based form are subsumed by the optimal reward framework—depending on the agent, environment, and the designer's objective return,

a reward function selected by the optimal reward problem may be of the potential-based form. Furthermore, in Chapter 6, I demonstrate that for some agents, potential-based shaping rewards are suboptimal.

2.3.2 Intrinsically Motivated Reinforcement Learning

Psychologists distinguish between *extrinsic motivation*, when an agent does something for a specific rewarding outcome, and *intrinsic motivation*, when an agent does something "because it is inherently interesting or enjoyable" (Ryan and Deci, 2000). Motivated by human traits such as curiosity and boredom, a growing number of agent designers have used modifications to the reward function to directly motivate behaviors which are auxiliary to an externally defined goal. These works generally demonstrate that the additional motivations improve performance with respect to an externally defined task, though a number tout other benefits, such as improved cognitive abilities which may be used in future tasks. As opposed to shaping rewards, the reward functions in this section tend to be agent-centric, environment-independent reward functions based on estimates of model quality, measurements of learning progress, drives to build knowledge, or others.

Schmidhuber (Schmidhuber, 1991a,b, 1997, 1999, 2005) introduced methods for implementing forms of curiosity using the reinforcement learning framework. These methods directly reward events such as reduction in error of the agent's model of the environment. He shows that such a signal can improve performance with respect to an externally defined task as well as lead to faster model learning.

Sutton (1990), in his work on the Dyna architecture, presented an experiment in which the agent was placed in a dynamically changing environment, even though the agent's model assumed that the environment was Markov and had stationary dynamics. To encourage the agent to discover areas in which the world had changed, he gave the agent a reward function that encouraged taking actions that had not been taken recently.

Duff (2003) developed an "Optimal Probe" which was driven by the objective of attempting to learn the environment model as quickly as possible. The reward function motivated the agent based on a measure of the agent's uncertainty in its model for a state-action pair: the variance of its Bayesian posterior distribution.

Barto et al. (Barto et al., 2004; Singh et al., 2005) describe a system for explicitly rewarding an agent for the development of hierarchical skills in the form of options (Sutton et al., 1999). They show that adding a motivation to learn skills can lead to

faster skill learning in a structured environment.

Şimşek and Barto (2006) derive a reward bonus proportional to the change in the agent's value estimate of the current state, effectively motivating the agent to learn to improve its value function estimates. They show that this can lead to an effective exploration strategy if the agent is given a period of exploration in which it is not being evaluated—in this dissertation's terminology, if the designer's objective return function is indifferent to the agent's behavior prior to some fixed horizon.

Oudeyer et al. (Oudeyer and Kaplan, 2007; Oudeyer et al., 2007) provide a number of different potential motivating signals which are unrelated to any external goal. They connect the resulting artificial behavior with natural human development. Though the behavior is not evaluated with respect to any specific task, they demonstrate that this behavior leads to sensible exploration strategies, where the most learnable stimuli are learned first, before the agent moves on to more complex stimuli.

In summary, the intrinsic motivation approaches encompass a variety of agent types and a variety of reward function modifications. For the most part, the reward functions in the IMRL line of work were designed to accelerate learning, although some were developed for an auxiliary purpose, unrelated to the agent's immediate performance. Although initial progress in the direction represented by these approaches has been promising, it has lacked a consistent analysis, because it has lacked the clarity of the separation of the agent's goals from the designer's goals.

Summary and Comparing IMRL to the Optimal Reward Problem. The IMRL line of work can be seen as a precursor to the optimal reward problem. Compared to the optimal reward problem, the IMRL work lacks the clarity of the separation of goals from agent and designer. The optimal reward problem promises to add a consistent, formal methodology to developing and analyzing approaches developed in the IMRL work. The optimal reward problem definition is general enough to encompass the reward functions in the IMRL literature. The definition of the designer's objective already accounts for the agent's performance and can be easily extended to measure other benefits of reward design such as the quality of its learned knowledge. Although intrinsic and extrinsic motivation have traditionally been viewed as distinct, Singh et al. (2010) use the optimal reward problem to argue that the distinction between extrinsic and intrinsic motivation is a matter of degree.

2.3.3 Reinforcement Learning with Sample Complexity Guarantees

A number of agent designers have developed reward functions while attempting to design agents with formal learning guarantees. The reward functions in this section have all been developed with a narrow purpose: accelerate learning. They do so by explicitly motivating exploration.

One of the simplest types of RL problems is known as an "n-armed bandit" problem, a generalization of a "one-armed bandit", or slot machine. This type of problem has received extensive attention from the learning theory and RL communities, because it captures an important aspect of an RL problem. At each step, the agent must choose whether to exploit its experience so far—pull the arm that has provided the highest return so far—or explore other arms in hopes that one provides a better expected return in the long run. This is known as the exploration–exploitation dilemma.

The UCB algorithm (Agrawal, 1995; Auer et al., 2002) handles this dilemma by explicitly motivating the agent to explore. The motivation appears in the form of an additive exploration bonus that can be interpreted as a reward for exploring. The reward bonus reduces as the the agent gathers more experience for that arm, causing the agent to exploit more often as it becomes more sure of its value estimates. The resulting agent achieves logarithmic regret as a function of the agent's lifetime, where regret is the difference in expected objective return between the agent's behavior and the optimal behavior.

In the general RL setting of sequential decision problems, designers have built agents which modify the reward function with a different type of theoretical guarantee. Kearns and Singh (2002) developed an agent which provably achieves high performance in a polynomial number of time steps with high probability in a general MDP setting. The type of result they showed is known as a PAC (Probably Approximately Correct) result. The method works by rewarding the agent for exploring states it has not visited sufficiently often to have learned a good model of them.

A number of similar PAC results followed, many of which also modify the reward function, including Rmax (Brafman and Tennenholtz, 2001), MBIE-EB (Strehl and Littman, 2008), approximate Bayesian planning (Kolter and Ng, 2009), and others (Asmuth et al., 2008). In all these works, the reward function explicitly motivates the agent to explore, generally to ensure the agent has sampled each action enough times to ensure it has an accurate model. Unlike in reward shaping, these reward functions explicitly result in different optimal policies than does the objective reward function. However, the reward functions in PAC methods are designed to converge to the agent

designer's goals.

Although not all PAC methods are model-based (or reward-based) (Strehl et al., 2006a), to the best of my knowledge, all existing reward-based methods with PAC guarantees are model-based. Implicit in these methods is the requirement that the agents have enough computational resources to compute the optimal policy with respect to the reward function they use. In other words, the rewards which come with PAC guarantees make strong assumptions about the computational capabilities about the agents they are used in.

Summary and Comparing PAC Approaches to the Optimal Reward Problem. The rewards in the PAC literature are similar to the rewards from the IMRL literature, in that they focus on properties of the agent's experience (such as how many times an action has been sampled) rather than on environment-dependent quantities, as does the reward shaping work. However, unlike the IMRL literature, the rewards in this section come with theoretical guarantees, and focus only on one aspect of agent design: the exploration–exploitation dilemma.

Much like in the previous cases, the reward functions in this section may or may not be solutions to a particular optimal reward problem. The theoretical guarantees that accompany the reward functions in the PAC literature are complementary to an optimal reward result. The theory involved can be useful in the development of reward features for use in an optimal reward problem. For example, in Chapter 4, I develop a novel reward with a PAC-like sample complexity result. I more thoroughly review related PAC literature in Chapter 4.

2.3.4 Other Work that Solves the Optimal Reward Problem

Singh et al. (2009, 2010) focus on the optimal reward problem in the context of natural agents, where the optimization problem is solved through evolution rather than by an agent designer, and the objective return obtained by an agent is replaced by its fitness in the environment. Niekum et al. (2010) take cues from this view and solve the optimal reward problem using genetic programming. Finally, there are a couple of works which use similar evolutionary approaches to reward optimization without explicitly acknowledging the separate notions of goals inherent in the optimal reward problem (Uchibe and Doya, 2008; Meriçli et al., 2010).

2.3.5 Summary of Reward Design Approaches

There is a long history of reward modification in the reinforcement learning literature. This history encompasses a variety of agent configurations and reward modifications. The optimal reward problem is general enough to encompass existing approaches. In other words, many of the reward functions proposed in related work above may indeed be solutions to particular instances of the optimal reward problem. The optimal reward problem is not an attempt to replace the above lines of work, but is instead an attempt to unify and better motivate these approaches.

Finally, I note that the perspective advanced in the next chapter onwards—that reward design compensates for agent limitations—is a broad one that extends beyond the scope of existing work, including other work on the optimal reward problem. Previous approaches have not taken advantage of the ability of rewards to compensate for general limits on agent design that prevent the agent from ever learning the optimal policy. In real agent design scenarios, this will often be the case—the real world is too large to model or learn exactly.

2.3.6 Natural Agents

The study of motivation in natural agents has a long history. Because the study of natural agents is heavily influenced by evolutionary theory, there are countless literature examples in various fields ranging from psychology to behavioral economics which discuss the optimization of preferences in natural agents. I discuss one example in this section which has been influential to this dissertation.

Samuelson and Swinkels (2006) analyze why "human utility embodies a number of seemingly irrational aspects." For example, many people are "irrationally" afraid of snakes, in the sense that they continue to be afraid even after being told that a particular snake is not dangerous. They argue that natural agents have properties such as this because evolution has compensated for the agents' limited abilities at processing information. People are not perfect at differentiating types of snakes, for example, and once a person learns whether his knowledge is correct, there is a chance he is dead. To compensate, evolution has programmed people to be afraid of snakes in general. The concepts I present in the next chapter, that optimal rewards are beneficial when agents are computationally limited, are the computational analog of this idea.

2.4 Chapter Summary

In any agent design problem there is both an agent and a designer, each with their own goals which can be represented using reward functions. In this dissertation, I consider the agent's reward function to be a parameter of agent design to be chosen by the agent designer. This results in the optimal reward problem.

This chapter discussed several foundational observations about optimal reward functions. The Fish-or-Bait experiment demonstrated that agents with optimal reward functions outperform agents with the standard notion of reward. It also demonstrated that optimal reward functions depend on properties of the agent, the environment, and the designer's objective return and that optimal reward functions can be vastly different from the designer's objective reward function.

Finally, prior work on reward design has a relatively long history; however, it has lacked the clarity of the separation of the agent's goals from the designer's goals. The optimal reward problem provides a common framework for analysis that is general enough to encompass the reward functions in prior work.

Chapter 3

Mitigating Agent Bounds[1]

Confounding the agent's and designer's goals, i.e., assuming that their reward functions should be the same, is a powerful and simple constraint on agent design, and so there is a cost to violating this assumption in terms of increased complexity of the design process. It is therefore important to understand when and why violating this assumption may improve agent performance with respect to the designer's objective. This chapter takes the first steps towards formalizing and empirically supporting a novel claim about the conditions under which solving the optimal reward problem is beneficial: reward design can lead to improved performance in RL agents that are limited, computationally or otherwise, in their ability to optimize their reward. Stated another way: if an agent is not able to achieve good performance with respect to the designer's expected objective return when given the objective reward function directly, it may behoove the designer to give the agent a different reward function. It is not immediately obvious that this should be the case. An agent that is limited in its ability to maximize objective return will generally be limited in its ability to maximize other definitions of return as well.

The second, and equally important claim in this chapter is that the nature of the agent's limitations impacts the reward function that can be used to mitigate those limitations. This can be used as a refinement of the above directive of agent design. An agent designer should choose a reward function that takes into account the agent's limitations.

For example, if an agent has limited computational resources that prevents it from building a full planning tree, a good reward function will help correct the errors in planning that result from the missing information. Alternatively, if an agent has approximations in its state representation that cause it to form poor policies in some areas of the state space, a good reward function may punish visiting those areas.

[1]This chapter presents material from Sorg et al. (2010a).

The key for an agent designer is to identify the limitations inherent in the agent architecture and to design the reward to compensate for them. In the main body of this chapter I provide several empirical illustrations of designed rewards that mitigate specific agent limitations that arise naturally in current practice in the design of learning agents. These include limits on planning depth and limits on state or model representation.

The notion of overcoming agent limitations is quite general. For example, prior work has demonstrated that reward design can be beneficial for the purpose of accelerating learning. Under the hypothesis that there always exists an optimal learning algorithm that would be chosen if it were not computationally prohibitive—such as Bayes-optimal learning (Duff, 2002)—the idea of using rewards to overcome computational limitations generalizes the idea of using rewards to accelerate learning. I discuss this concept in more detail in Chapter 4, in which I derive a reward for directly approximating an optimal learning algorithm.

Finally, because this method of analysis focuses on limitations inherent in an agent architecture, which may be broadly applied to a wide range of environments, the rewards designed to mitigate the agent's limitations often may also be able to be applied to a wide range of environments. The reward functions designed in this chapter have this property: they depend on features of an agent's history that may be applied across multiple environments.

Specifically, I decompose reward functions into additive *reward features* and discuss and evaluate three general reward features for overcoming various types of agent limitations in several environments. One feature, *inverse-recency*, motivates an agent to persistently explore the environment by rewarding it to explore state-action pairs it has not tried recently. The agent's need to explore is created by its lack of ability to plan or perceive far enough to reach its goals. This feature is similar to the one used by Sutton (Sutton, 1990). I also discuss the $1/n$ *exploration bonus* (Kolter and Ng, 2009), which motivates transient exploration that fades as experience is gathered. This feature has been shown to approximate Bayes-optimal learning without increasing the size of the planning state space. Finally, I present the *local-model-inaccuracy* feature, which punishes the agent for visiting areas of the state space that it has trouble modeling.

The remainder of the paper is organized as follows. I first provide an ontology of bounded (limited) and unbounded agents and the *performance gaps* between them that will help to sharpen the claims above and provide a general framework in which to position the empirical results. The main body of the chapter then presents empirical

results demonstrating the claims, providing examples of the reward features above being used to improve the performance of agents with various limitations.

3.1 Reward Design is for Bounded Agents

Agents are typically defined via some parameters such as learning and exploration rates for Q-learning agents, planning depth for look-ahead-based planning agents, etc. In general, the designer must not only set these parameters, the designer must select the agent architecture itself. Let $\theta \in \bar{\Theta}$ denote a particular setting of all the parameters of an agent, including the choice of agent architecture, chosen from the set of all possible agent definitions $\bar{\Theta}$. Specifically, it corresponds to the set of all mappings from histories to distributions over actions, parameterized by reward function. I refer to $\bar{\Theta}$ as the *unbounded* set of agent parameters, because it is unconstrained. Thus, $G(R, \theta)$ fully specifies an agent in this work, where R is the reward function parameter to the agent and θ is all other parameters.

In practical situations, however, the designer searches a constrained subset of agents $\Theta \subset \bar{\Theta}$. This set is typically constrained by the set of agent architectures that he or she prefers based on prior experience. In this work, I focus on autonomous agents that seek to optimize their goals. This can be seen as one constraint on Θ. Most importantly, the designer's choice of θ is constrained by the computational resources available. For example, increasing the planning-depth in a look-ahead-based planning agent typically increases the agent's computational cost. I assume that the agent is being executed on a machine with fixed computational power. Thus, there are likely to be some values of the planning depth that are infeasible—for example, if the agent's memory resources are limited. The result is that a particular agent $\theta \in \Theta$ that the designer chooses will have limitations that an optimal agent chosen from the unbounded space $\bar{\theta}^* \in \bar{\Theta}$ won't have. Reward design helps to mitigate these limitations. I formalize these notions next, by defining and comparing a set of related agent optimization problems.

	Bounded Θ $\theta^* \in \Theta$	*Unbounded* $\bar{\Theta}$ $\bar{\theta}^* \in \bar{\Theta}$
Objective reward R_O	(A) Bounded-conventional $G(R_O, \theta^*)^\dagger$	(B) Unbounded-conventional $G(R_O, \bar{\theta}^*)^\ddagger$
Optimal reward $R^* \in \mathcal{R}$	(C) Bounded-optimal $G(R^*, \theta^*)^\S$	(D) Unbounded-optimal $G(R^*, \bar{\theta}^*)^{\text{❋}}$

$^\dagger\, \theta^* = \arg\max_{\theta \in \Theta} \mathbb{E}\left[U_{R_O}(h) | h \sim M\langle G(R_O, \theta)\rangle\right]$
$^\ddagger\, \bar{\theta}^* = \arg\max_{\bar{\theta} \in \bar{\Theta}} \mathbb{E}\left[U_{R_O}(h) | h \sim M\langle G(R_O, \bar{\theta})\rangle\right]$
$^\S\, (R^*, \theta^*) = \arg\max_{R \in \mathcal{R}; \theta \in \Theta} \mathbb{E}\left[U_{R_O}(h) | h \sim M\langle G(R, \theta)\rangle\right]$
$^{\text{❋}}\, (R^*, \bar{\theta}^*) = \arg\max_{R \in \mathcal{R}; \bar{\theta} \in \bar{\Theta}} \mathbb{E}\left[U_{R_O}(h) | h \sim M\langle G(R, \bar{\theta})\rangle\right]$

Table 3.1: Four variants of the optimal reward problem. Each agent class is constructed as a solution to an optimal reward problem defined by constraints on the reward function space (rows) and the agent's conventional parameter space (columns).

3.1.1 Comparing Agent Design Spaces

To sharpen the notion of performance differences between bounded and unbounded agents, it is useful to consider the four types of agents defined in Table 3.1[2]. Each agent, labeled by a letter in A–D, is defined as a solution to an optimal reward problem extended to include joint optimization over agent parameters. Note that all the agents are optimized with respect to the expected objective return function, *whether or not* they actually use the objective reward function internally.

The top row in the table contains agents that use the objective reward function R_O. The *bounded-conventional* agent $\text{A} = G(R_O, \theta^*)$ is the conventional design goal of an agent designer in an RL setting. It is the best agent a conventional agent designer will build given computational and other constraints, using the objective reward function. An *unbounded-conventional* agent $\text{B} = G(R_O, \bar{\theta}^*)$, in contrast, is the best agent in principle that uses the objective reward function. An unbounded-conventional agent is not guaranteed to satisfy the design constraints as specified by Θ.

The bottom row in the table contains agents that have designed reward functions. A *bounded-optimal agent* $\text{C} = G(R^*, \theta^*)$ is the best an agent designer could build given the design constraints; it is a solution to the joint optimization over the set $\mathcal{R} \times \Theta$. An unbounded-optimal agent $\text{D} = G(R^*, \bar{\theta}^*)$ is a solution to the joint optimization over the set $\mathcal{R} \times \bar{\Theta}$. There is no better agent than the *unbounded-optimal* agent D.

[2]Table 3.1 is identical to Table 1 in Sorg et al. (2010a), but the left column has been removed and the notation has changed.

The performance gaps among agents.

It should be clear from the definitions of the agents in Table 3.1 that the following partial ordering holds among the performances of agents: $A \preceq B \preceq D$ and $A \preceq C \preceq D$. This is because, comparing adjacent agents in the table, the agent to the right of (or beneath) the other involves an optimization problem with a search set that contains the search set of the optimization problem to the left (or above) the first.

There is no performance difference between B and D ($B \not\prec D$), because unbounded-conventional agents are optimal with respect to expected objective return. Thus, there is no performance gain to be had through reward design when the agent is not limited. The implications of this are important: as the set of available agents expands (Θ approaches $\bar{\Theta}$), the opportunity for reward design to help diminishes.

The performance difference or *gap* of primary theoretical interest in this chapter is the difference between the θ-optimal agent A and the θ-unbounded agent B. This is the loss in performance caused by the designer's constraints on Θ, compared to the best possible agent the designer could build if there were no constraints.

The central claim of this chapter is that for many environments and available parameters Θ, the performance of the boundedly-optimal agent C will be *strictly greater* than that of the θ-optimal agent A; that is, the agent C's performance falls in between the performances of the bounded agent A and the unbounded agent B, i.e., $A \prec C \preceq B$. Thus, optimal rewards mitigate the performance gap between bounded agents and unbounded ones.

The second claim made in this chapter is that the nature in which Θ is bounded—the nature of the constraints that prevent the designer from producing an unbounded agent—aid in designing reward functions to mitigate this gap. The empirical section provides examples to support this claim.

The focus of this work is not on optimizing θ. As such, the experiments do not solve this joint-optimization problem. Instead, the experiments focus on values of θ that are chosen to be representative of different types of agents used in practice. This is sufficient to demonstrate the claims in principle, as a given θ is a (bounded) singleton set. The experiments in the next section examine the manner in which the given θ is limited in relation to the optimal unbounded agent $\bar{\theta}^*$ and use this analysis to design reward functions to mitigate the resulting performance loss.

3.2 Empirical Demonstrations

This section presents four experiments designed to illustrate the mitigation of agent bounds by optimal rewards. Each involves a model-based learning agent with a bound on planning depth, state representation, or model representation. The agents are placed in simulated foraging environments.

Each experiment has the same structure:

1. I investigate the performance of an agent with some specific limitation in an environment that reveals the performance loss that arises from that limitation.

2. By examining the interaction between the agent's limitations and the environment, I design a set of (mostly environment-independent) features that are designed to overcome each limitation. I formulate a set of rewards as a linear mapping from these features to reward.

3. Through search of this set by directly simulating agents in their environments, I find an approximately-optimal reward $\hat{R}^*$ and compare the performances of the approximate bounded-optimal agent C $= G(\hat{R}^*, \theta)$, the bounded-conventional agent A $= G(R_O, \theta)$, and the unbounded-conventional agent B $= G(R_O, \bar{\theta}^*)$ (if available).

To get a sense of how an agent's limitations can affect the reward design problem, Experiment 1 does (1)–(3) while also varying the agent limitation parametrically.

3.2.1 Experiment 1: Mitigating bounds on planning depth

The objective of this experiment is to show that rewards can mitigate bounds on planning depth. Specifically, Experiment 1 shows that rewards based on inverse-recency features can mitigate bounded planning in environments that require persistent exploration. The experiment also provides an empirical demonstration that an unbounded agent may not benefit from breaking the confound.

The environment and designer's goals. Consider the foraging environment illustrated in Figure 3.1. The environment consists of a 3×3 grid world with 3 dead-end corridors separated by impassable walls. The agent, represented by the bird, has four available actions that deterministically move the agent in each of the cardinal directions. If the intended direction is blocked by a wall or the boundary, the action

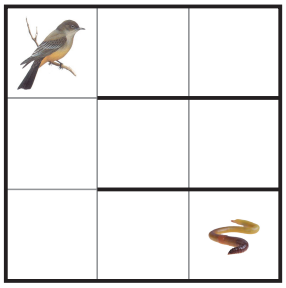


Figure 3.1: Foraging Environment

results in no movement. There is a food source, represented by the worm, randomly located in one of the three right-most locations at the end of each corridor. The agent has an `eat` action, which consumes the worm when the agent is at the worm's location. After the agent consumes the worm, the agent becomes *satiated* for 1 time step, and the worm disappears. Immediately, a new worm appears randomly in one of the other two potential worm locations. At all other time steps, the agent is *hungry*. The agent observes the entire state: the agent's location, whether it is hungry, and the worm's location.

The designer's goal is to maximize the number of worms eaten—or equivalently, the fraction of time steps in which the agent is satiated. (In this experiment, as opposed to Bait-or-Fish in Chapter 2, the designer fully values eating worms.) Thus, the objective reward function R_O provides a reward of 1.0 when the agent eats a worm (i.e., is satiated) in the current observation, and a reward of 0 otherwise.

In this chapter, I focus exclusively on the infinite-horizon average reward accumulation function (as in Section 2.1). In this setting, the agent designer does not care about how quickly an agent learns—it only cares about the asymptotic performance of the agent. Thus, the experiments here focus exclusively on computational bounds that are detrimental to asymptotic performance, ignoring the effects of reward modification on the rate of learning. This allows the results to focus solely on the novel claim of benefit from rewards design in this chapter, that rewards mitigate agent boundedness.

The agent and its limitations. The agent plans using a learned model of the transition dynamics. The estimates, $\hat{T}$, come from an estimated MDP transition model (updated after every action) based on the empirical transition probabilities between assumed-Markov observations. Specifically, let $n_{o,a}$ be the number of times that action a was taken in state o. Let $n_{o,a,o'}$ be the number of times that o' was reached after taking action a in state o. The agent models the probability of reaching

o' after taking a in state o as $\hat{T}(o'|o,a) = \frac{n_{o,a,o'}}{n_{o,a}}$.[3]

The general form of the rewards in all the experiments is $R(h,o,a,o';\beta) = \beta^{\mathsf{T}}\phi(o,a,o',h)$, where β is a parameter vector, and ϕ is a vector of features that may depend on current and next observations o and o'; the action a; and the history h prior to the current observation.

Let G_d (short for $G(R,d)$) denote a *Full Finite-Horizon Planning (FFHP) Agent*—an agent that acts greedily with respect to the H-step action-value function

$$Q^H(o,a) = \sum_{o' \in \mathcal{O}} \hat{T}(o'|o,a)[R(h,o,a,o') + \gamma \max_{a'} Q^{H-1}(o',a')], \tag{3.1}$$

where $Q^0(o,a) \stackrel{\text{def}}{=} 0$. The agent computes these Q-function estimates at each step, and acts by taking the greedy action. If the values of multiple actions are equivalent, the agent selects randomly among them. Note that the history h affects the reward function, but does not appear in the transition model or the Q-function estimates. In other words, the designed reward function is allowed to change each step based on the agent's experience prior to the current time step, but the agent's future plans still make the Markov assumption on observation. The reward functions proposed below are examples of reward functions that have these properties. The agents use $\gamma = 0.99$ unless noted otherwise[4].

The agent G_d is a simple example of a computation-bounded agent in which the depth H is a parameter controlling the degree of boundedness. More specifically, agent G_0 is a random agent, because its Q-function is a constant 0; agent G_1 acts greedily with respect to its reward, and agent G_∞ is an unbounded-depth planning agent, computing the optimal value function with respect to its current model and internal reward function.

In experiment 1's environment, the largest (over all states) look-ahead needed to obtain objective reward is 8. Thus I explore agents with planning depths between 0 and 9, where G_8 and G_9 are equivalent to G_∞. Crucially, it is the inability of agent G_H for $H < 8$ to encounter an objective reward during planning from some states that I wish to mitigate via reward design.

[3]Before an observation-action pair is experienced (i.e., when $n_{o,a} = 0$) the transition model is initialized to the identity function: $\hat{T}(o'|o,a) = 1$ iff $o' = o$.

[4]When planning with bounded depth, using $\gamma < 1$ encourages the agent to pursue rewards more quickly, rather than waiting to obtain them at the horizon. Note that this implies that the agent's accumulation function uses discounting while the designer's accumulation function does not. Using $\gamma < 1$ to achieve this effect is another instance of agent goal design, though not one this work focuses on.

Reward Design Space. The choice of features for this domain is driven by the following intuition. If an agent G_H is more than H steps away from the worm, for $H < 8$, what action should it take? The agent could take random actions to explore randomly to achieve a state that is within H steps from the worm, but this will be inefficient. A good reward function would lead to some kind of systematic and persistent exploration and would thus be far more efficient.

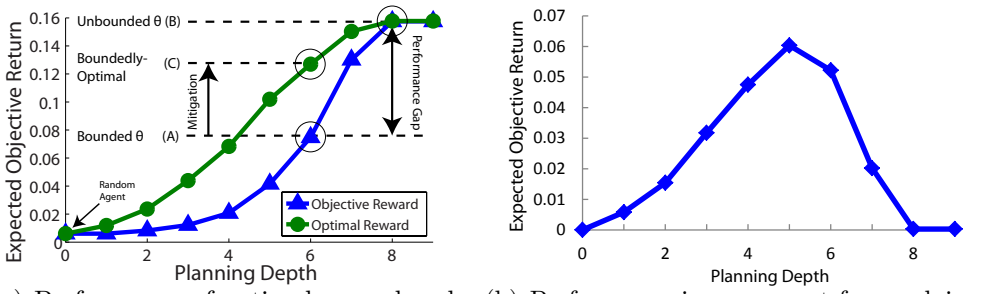
Specifically, I consider the reward set $R(h, o, a, s', \beta) = \beta_O \phi_O(o) + \beta_c \phi_c(ho, a)$, where β_O and β_c are the two parameters, feature $\phi_O(o)$ is 1 when the agent is satiated in state o and 0 otherwise, and feature $\phi_c(h, o, a) = 1 - \frac{1}{c(h,o,a)}$, where $c(h, o, a)$ is the number of time steps since the agent previously executed action a at observation o within history h. Feature ϕ_c captures inverse-recency; the feature's value is high when the agent has not taken the indicated observation-action pair recently and is low when the agent has taken it recently. When β_c is positive, the agent is rewarded for taking actions that it has not taken recently from the current observation. Note that when $\beta_O = 1$ and $\beta_c = 0$, the reward function equals the objective reward function.

Intuitively, positively rewarding the inverse-recency feature encourages persistent exploration. Of course, another method of persistent exploration is by taking random decisions. This experiment demonstrates that if $H < 8$, the use of an inverse-recency reward can improve performance, but when a planning depth $H \geq 8$ is used, the agent does not benefit from reward design.

Results

I approximately solved the optimal reward problem for each depth $H \in \{0, 1, \cdots, 9\}$. This parameter sweep is best viewed as varying the computational limits available to the agent designer, because a designer will generally choose the maximum depth allowable by the constraints. I then evaluated the use of the objective reward function R_O and the approximately optimal reward $\hat{R}^*_d$ at each depth H for 200,000 steps, averaged over 200 trials, to estimate the expected infinite-horizon objective return of each reward function.

The optimal reward problem was approximately solved by measuring the mean objective return obtained during a 10,000 step horizon. The optimization procedure in this chapter adaptively samples reward vectors from a discrete set on the unit sphere, as it can be shown that for the (linear) form of the reward functions and for the agents presented here, searching the unit sphere is equivalent to searching the entire space. Although the procedure was not guaranteed to find an optimal reward, the claim



(a) Performance of optimal reward and objective reward as a function of depth

(b) Performance improvement from solving the ORP, as a function of depth

Figure 3.2: Results from Experiment 1 on limited-depth planning, showing performance gains from solving the ORP as a function of the planning depth bound.

that reward mitigates agent limitations is demonstrated by simply finding any reward function that outperforms the objective reward function for the target agents.

As can be seen in Figure 3.2, the approximately optimal reward function $\hat{R}^*_d$ performs at least as well as the objective reward function at all planning depths. When the planning depth is 0, both agents do not plan and simply act randomly. When the planning depth is 8 or more, the agent is in effect unbounded for this environment, the objective reward acts optimally, and no other reward can do better. Between these two extremes, however, the use of designed reward greatly benefits the agent. In some cases, $G(\hat{R}^*_d, d)$ performs as well or better than $G(R_O, d+2)$.

These results are consistent with a general pattern that might be expected to hold across other kinds of agent bounds and environments: the benefit of reward design reaches its maximum at some intermediate level of boundedness, but approaches zero as the agent either approaches the unbounded agent at one extreme, or a degenerate (perhaps random) agent at the other extreme. This trend is illustrated in Figure 3.2(b), which plots the difference in performance between the approximately optimal reward and the objective reward as a function of depth.

3.2.2 Experiment 2: Bounds on state representation

The objective in Experiment 2 is to show that rewards based on the same inverse-recency features as used in Experiment 1 can mitigate a different kind of agent limitation, that of impoverished state representation.

The environment and designer's goals. Identical to Experiment 1, with one exception. In contrast to Experiment 1, the agent observes only its location, whether or not it is hungry, and whether or not it is colocated with the worm; the agent *cannot* see the worm unless it is colocated with it.

An unbounded agent in this situation would use a state representation that takes into account the history of the agent's interaction. For example, by tracking the locations that the agent has visited since the last time it ate, the agent can figure out which locations have a chance to contain the food, and use this information to its advantage. Representations such as the belief state representation (Kaelbling et al., 1998) handle this partial observability optimally. However, the size of the belief state space is often much larger than the original state space, requiring more computational resources.

The agent and its limitations. The agent is an unbounded-depth planning model-based learning agent G_∞. The agent implements G_∞ using value iteration that terminates after the max change in value falls bellow the threshold $\delta < 10^{-4}$. Action values are considered equal if they differ by less than this threshold.

The decision process faced by the agent is not an MDP. Nevertheless, to reduce the computational requirements of the agent, it uses the same model-learning procedure as was used in the previous experiment (the agent continually updates an estimated MDP model as if observations were Markov state). Given that the agent cannot observe the location of food, it cannot plan to reach the food in the shortest path, even with its infinite look-ahead. Also, given that the agent's observations do not tell it what locations it has visited since the last time it ate food (and hence should be known to be empty locations), it cannot plan to avoid exploring locations that should be known not to have food based on the agent's history. In fact, the objective reward agent does quite the opposite; it tends to get stuck attempting to explore the same location repeatedly.

The reward design space. I use the inverse-recency reward space, as described above in Experiment 1. The intuition here is similar to the intuition in Experiment 1. The effect of the partial observability is that the agent can't accurately plan deep into the future. This effect is not unlike the distortion in value calculations caused by the horizon in the depth-limited planning agent.

Agent type	β_O	β_c	$\mathbb{E}[R_O]$ per step
Random	0	0	0.0060 ± 2.46e-5
Conventional; R_O	1	0	8.6e-6 ± 4.55e-7
R-Optimal; $\hat{R}^*$	0.147	0.989	0.0745 ± 2.15e-4
Unbounded	N/A	N/A	0.1153 ± 2.90e-5

Table 3.2: Results from Experiment 2 on the Bounded-State Agent in the Foraging Environment.

Results

In Table 3.2, I compare the agent G_∞ using the objective reward function R_O with the agent using the approximately optimal reward $\hat{R}^*$. Each estimated value is the mean objective utility obtained per step over $200,000$ steps, averaged over 200 trials. As predicted, the optimal reward outperforms the objective reward.

This table also shows the specific values of reward parameters. The optimal reward function positively rewards eating and the inverse-recency feature. Noteworthy is the relatively large coefficient for the inverse-recency feature relative to the coefficient for the confounded-reward feature. This is because of the errors in the model caused by the partial observability. Not only does the inverse-recency reward need to encourage exploration, it needs to overcome the errors in the agent's model that predict that food might be nearby, even though the agent's history indicates otherwise.

Additionally, I compare against two reference agents: a random agent, and an unbounded belief-state agent. The unbounded belief-state agent learns a POMDP model that helps the agent keep track of which potential food locations it has visited since the most recent time it ate, and it accounts for changes in belief during planning. As can be seen, the optimal reward performs much better than the objective reward, much better than a random agent, and not quite as well as the unbounded belief-state agent. The optimal reward manages to achieve this despite being coupled with a model that is wholly inadequate at predicting the food location.

3.2.3 Experiment 3: Bounds on model representation

The objective of Experiment 3 is to introduce the *local-model-inaccuracy* reward feature and show that it can mitigate boundedness arising when an agent cannot model its environment uniformly accurately. The experiment also uses the $1/n$ exploration bonus (Kolter and Ng, 2009), which encourages exploration that fades as the agent gathers more experience— I refer to this as *transient exploration*.

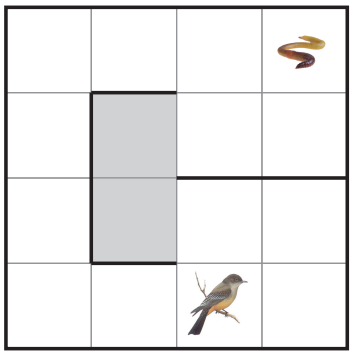


Figure 3.3: Dark Room Environment

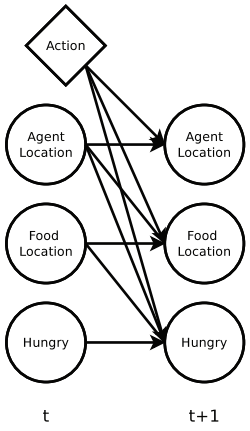


Figure 3.4: Independence Assumptions in Experiments 3 & 4.

The environment and designer's goals. Figure 3.3 illustrates the Dark Room environment. The worm is located in either the top-right or bottom-right corners. As before, the agent's goal is to eat the worm, after which it becomes satiated for 1 time step. After the worm is eaten, a new worm appears in the other right corner. Unlike Experiments 1 and 2, the agent's movement is stochastic—each movement action fails with probability 0.1, resulting in movement in a random direction. The environment has been modified in this way to make it more difficult to model.

The special feature of this world is the dark room in the center which spans two locations. The agent's location sensor cannot distinguish between the two locations inside the dark room but works perfectly outside the dark room. The agent always perceives the location of the food and whether it is hungry.

The agent and its limitations. The agent is an infinite-depth planning model-based learning agent (G_∞). The agent assumes a *factored* model of the three observation variables—Agent Location, Food Location, and Hungry. Figure 3.4 depicts a graphical model of the factored assumptions that were designed to compactly but accurately capture the structure in the original fully-observable 3-Corridor foraging environment of experiments 1 & 2. However, the Dark Room environment violates the agent's Markov assumption for the location variable—in the dark room, the agent's next location observation is not independent of the history given its current location observation. The agent learns the probabilities of each component of the factored model using empirical counts, as was done for the unstructured modeling agents above.

The reward design space. The reward feature space in Experiment 3 includes two new environment-independent features—the local-model-inaccuracy feature and the $1/n$ exploration bonus.

The first is a feature that estimates the *quality* of the agent's model as a function of observation. The intuition is that such a feature might be exploited by the optimal reward function to motivate the agent to avoid areas it has difficulty modeling. Specifically, the local-model-inaccuracy feature ϕ_δ, keeps a simple moving average of the recent errors in predicting the next observation. In this factored case, it tracks the errors in predicting each observation feature and then averages them. Let o_i denote the value of observation feature i in observation o, and let $\pi(i)$ denote the set of features that feature i depends on in the previous observation. The agent's transition model is $\hat{T}(o'|o,a) = \prod_i \hat{T}_i(o'_i|o_{\pi(i)},a)$. For the model-error feature, I keep a parameter $\rho_{o_{\pi(i)},a,o'_i}$ for each unique $\langle o_{\pi(i)}, a, o'_i\rangle$ tuple. After observing a transition from observation o to observation o', each component model has instantaneous error $\delta = 1 - \hat{T}_i(o''_i|o_{\pi(i)},a)$ if $o''_i = o'_i$ and $\delta = -\hat{T}_i(o''_i|o_{\pi(i)},a)$ if $o''_i \neq o'_i$. Each error parameter matching the conditioned observation and action features is then updated according to $\rho \leftarrow \rho + 0.1(\delta - \rho)$, where each ρ is initialized to 0. These errors are then averaged across component models: $\phi_\delta(h,o,a,o') = \frac{1}{3}\sum_i (\rho_{i,o_{\pi(i)},a,o'_i})^2$.

For a correct model, the expected value of the ρ in the model-error feature is zero. However, it will tend to fluctuate around 0 in a stochastic environment. It is for this reason that the foraging environment has been modified to include stochastic movement—to show that the reward can distinguish between stochastic dynamics and an erroneous model.

One challenge in designing a model-building agent that is motivated to avoid states in which it has a bad model is that it will start with a bad model. To motivate the

Agent Type	Reward parameters β_O	β_δ	β_n	Dark Room *mean R_O per step*	Windy *mean R_O per step*
Random	0	0	0	0.0044 ± 1.67e-5	0.0018 ± 3.03e-5
Conventional, R_O	1	0	0	0.0138 ± 2.43e-3	0.0335 ± 4.56e-3
R-Optimal (Dark Room), $\hat{R}^*$	0.267	-0.963	0.047	0.0895 ± 7.82e-5	0.0782 ± 3.18e-4
R-Optimal (Windy), $\hat{R}^*$	0.219	-0.975	0.048	0.0896 ± 4.38e-5	0.0806 ± 2.15e-4
Rounded, R	0.25	-1	0.05	0.0896 ± 6.89e-5	0.0801 ± 1.73e-4
Optimal (avoiding trap states)	N/A	N/A	N/A	0.0903 ± 1.23e-5	0.0826 ± 9.34e-6

Table 3.3: Factored Model Agent in Dark Room and Windy Environments. (Experiments 3 & 4.)

agent to build its model in the first place, I provide an additional reward feature $\phi_n(h, o, a)$ which is inversely proportional to how often action a has been taken in state s. Specifically, $\phi_n(h, o, a) = \frac{1}{3}\sum_i \frac{1}{n_{o_{\pi(i)},a}+1}$, where $n_{o_{\pi(i)},a}$ is the number of times action a was taken when the features in the parent set of i matched $o_{\pi(i)}$ according to history h. A positive coefficient for this feature encourages the agent to experience state-action pairs it hasn't experienced often. The feature is by definition transient and its magnitude decreases steadily with increased experience.

In summary, the reward space provides to the optimal reward problem is defined by $R(h, o, a, o', \beta) = \beta_O\phi_O(o) + \beta_\delta\phi_\delta(h, o, a, o') + \beta_n\phi_n(h, o, a)$ for some parameters β_O, β_δ, and β_n. Motivating the learning of a model is perhaps the original view of the purpose of reward design in model-based agents (e.g., Brafman and Tennenholtz, 2001; Strehl and Littman, 2005) In recent work, Kolter and Ng (2009) showed that adding reward proportional to $1/(n_{o,a} + 1)$, where $n_{o,a}$ is defined as above, to the objective reward approximates the behavior of an unbounded Bayes-optimal learner. I discuss this result further in Chapter 4.

Results

Table 3.3 presents the performance of the factored-model agent in the Dark Room environment (and the Windy World environment, described next). Each estimated value is the mean objective utility obtained per step over $200,000$ steps, averaged over 200 trials. I tested the factored agent with the following set of rewards: $R \stackrel{\text{def}}{=} 0$ (this corresponds to random behavior), the objective reward R_O, and the approximately optimal reward $\hat{R}^*$ found through optimization. In addition, I present the performance of the best agent that avoids the difficult-to-model (trap) state(s) as an approximation

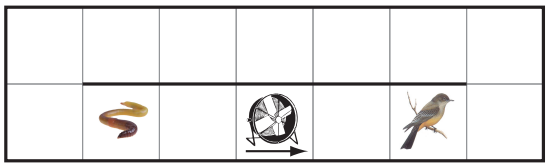


Figure 3.5: Windy World Environment.

of the optimal agent[5].

The optimal reward significantly outperforms the agent with the objective reward. The latter is unable to learn to navigate up or down in the dark room; instead it repeatedly enters and exits the dark room on the same side, expecting to sometimes appear on the other side. The optimal reward function motivates eating, strongly punishes visiting locations that have high model inaccuracy, and lightly rewards exploring. The local-model-inaccuracy feature successfully detects the agent's limitations at predicting state transitions in and out of the dark room. By punishing the agent for experiencing these state transitions, the optimal reward function encourages the agent to avoid areas of the state space it has trouble modeling and to take the path that is easier to model.

3.2.4 Experiment 4: Bounds on model representation, and an environment-independent reward function

The approximately optimal reward found in Experiment 3 is the result of a careful tuning of the reward coefficients β_O, β_δ, and β_n. Although the sign of these coefficients (positive, negative, and positive, respectively) is consistent with expectations, their precise values may be determined largely by very specific properties of the domain. It is therefore interesting to begin exploring how effective such specific rewards are when transferred to related but distinct domains, and this is one of the key objectives of Experiment 4.

The environment. Figure 3.5 illustrates the Windy World environment. It shares the basic properties of the Dark Room environment—stochastic movement, unique location identifiers. The food source is in one of two locations, either the location

[5]In the Dark Room domain, the unbounded agent is the belief-state agent, which would likely choose to navigate the dark room, but I chose not to implement this agent. The optimal agent that avoids the dark room provides a lower-bound on the true unbounded agent's performance.

pictured, or the location currently occupied by the agent. Instead of a dark room, however, this world has a different twist—there is a giant fan in the bottom-middle location. At all times, the fan directs a forceful wind in the direction *away* from the worm. If the agent is in the fan's location, its action will fail with probability 0.95 and the agent will transition to the state in the direction opposite from the worm. Otherwise, the attempted movement proceeds as normal, with a 0.10 chance of failing as in other locations.

The agent and its bounds. The agent is identical to that in Experiment 3. The Dark Room environment violated the Markov assumption in the factored model for the location variable but all other independence assumptions were accurate. The Windy World environment, on the other hand, violates the assumption that the agent's next location is independent of the current location of the worm—when co-located with the fan, the next location is correlated with the worm's location. The factored agent with the objective reward attempts to push its way through the fan, though it is more efficient to walk around the long way.

The reward space. The reward space in Experiment 4 is identical to the reward space in Experiment 3.

Results

Table 3.3 presents the results for the Windy World environment. As in Dark Room, the agent with the optimal reward significantly outperforms the conventional agent. Table 3.3 also shows the performance of the agent in the Dark Room world when provided with the approximately optimal *Windy World* reward, and vice versa. I also tested a single reward function derived by simply rounding off the β-coefficient values found optimal in both the Dark Room and Windy environments. All of the optimized rewards, including the rounded reward, perform well in both environments, significantly outperforming the conventional agent and nearly closing the gap to the optimal agent (that takes the long way around). This is a small and preliminary illustration that the practice of reward design based on domain-independent features may yield reward functions that are robust across different environments.

3.3 Discussion

This chapter presented a novel motivation for the use of reward design. Critically, and as a departure from prior work, the reward functions presented here were designed with the explicit intention of modifying the agent's behavior in such a way that it produces different behavior, both in the short-term and long-term, than the same agent would exhibit when using the objective reward function.

The empirical results validated the two claims stated in the introduction: (1) rewards can be used to overcome agent boundedness; and (2) agent designers can design good reward functions by analyzing the manner in which their agents are bounded, reasoning about the way this limitation interacts with the environment, and designing rewards to compensate. In this chapter, the reward features—*inverse-recency*, *local-model-inaccuracy*, and $1/n$—were suggested heuristically using an intuitive understanding of the issues. In the next chapter, I show that reward features can be derived theoretically, such that strong properties hold for particular values of the reward function coefficients β.

Chapter 4

Deriving a Reward Function for Approximately Optimal Learning[1]

One of the central challenges of reinforcement learning is the explore–exploit dilemma. An agent must maximize its expected reward (exploit) while simultaneously sacrificing immediate gains to learn about new ways to exploit in the future (explore). In one approach to addressing the dilemma, Bayesian Reinforcement Learning, the agent is endowed with an explicit representation of its uncertainty in the form of a distribution over the environments it could be in. As it acts and receives observations, it updates its belief about the environment distribution accordingly. A Bayes-optimal agent solves the explore–exploit dilemma by explicitly including information about its belief in its state representation and incorporating information changes into its plans (Duff, 2003). However, Bayes-optimal planning is intractable in general. A number of recent methods have attempted to approximate Bayesian planning (Poupart et al., 2006; Asmuth et al., 2009; Kolter and Ng, 2009), but this remains a challenging problem.

Another approach to addressing the explore–exploit dilemma is to assign the agent a reward function which explicitly rewards exploring. An agent that always exploits this reward function accomplishes both exploration and exploitation (with respect to the original objective reward function). This approach is exemplified by many approximate methods in the PAC framework (Kearns and Singh, 2002; Strehl et al., 2006b; Strehl and Littman, 2008; Kolter and Ng, 2009) which bound the complexity of learning an MDP by explicitly motivating the agent to sample state-action pairs enough times to ensure it has explored sufficiently. Reward-design methods have one advantage over the Bayesian approach: modifying the reward function can greatly influence behavior, often without greatly affecting computational cost.

In contrast, an important advantage of the Bayesian approach is that exploration is guided by prior knowledge of the environment. Some environments may require

[1]This chapter presents material from Sorg et al. (2010c).

more exploration than others; some areas of the state space may be more uncertain than others; and most interestingly, information gained in one area of the state space may affect knowledge about other areas. The Bayesian approach expresses all of these through the specification of the agent's prior belief.

In this chapter, I contribute a reward feature for motivating exploration which is based on knowledge in the form of an agent's Bayesian posterior distribution over environments. Rather than performing full Bayesian planning, which is intractable, the agent I use in this chapter approximates Bayesian planning by planning in the much smaller *mean MDP*. This mean MDP approximation performs poorly when the agent plans with the objective reward. To compensate for this limitation, I present the *variance-based reward bonus*, which is based on the variance of the agent's posterior belief distribution over environment models. When using this reward bonus, scaled appropriately, the agent explores efficiently—with provable sample complexity bounds similar to the PAC methods referenced above. In other words, the variance reward bonus mitigates the agent's planning limitation caused by the mean MDP approximation.

Compared to existing reward-based PAC methods, the proposed variance-based reward bonus is able to take advantage of the knowledge contained in the prior distribution. The variance-based reward bonus is similar to existing reward bonuses when using an uninformative prior, such as the independent Dirichlet distribution over transition dynamics. For informative priors however, I show that it is capable of achieving a theoretically lower sample complexity bound than existing approaches. Finally, I demonstrate empirically that the variance reward approach compares favorably to existing reward bonuses when using both structured and unstructured priors in two environments, including the Hunt the Wumpus Environment (Russell and Norvig, 2002), where prior knowledge is critical and over-exploration can lead to permanent death.

4.1 Mean MDP Planning and the Variance-Based Reward Bonus

An MDP is the tuple $M = \langle S, A, R_O^M, T^M, \gamma \rangle$, where S is the state space, A is the action set, $R_O^M(s,a)$ is the objective reward function of MDP M, $T^M(s'|s,a)$ is the probability of transitioning to state s' given that action a was taken in state s in MDP M, and γ is the discount factor. Sometimes, when it is clear from context, I use S

and A to also refer to the cardinality of the corresponding sets. For the theoretical results in this chapter only, without loss of generality, I assume that the objective reward function is in the range $[0, 1]$.

4.1.1 Bayes-Optimal Planning

Unlike the standard RL setting, in Bayesian RL the agent is provided a-priori with the distribution of environments it could face, called the *prior* or the initial *belief* b_0. For belief b, I denote the probability of a particular MDP M as $b(M)$. When the agent begins acting, it does not know which specific environment it is in, but instead updates its belief based on experience. After observing a state transition from s to s' and reward r, the agent updates its belief b to the Bayesian *posterior* belief b' using Bayes' rule. Specifically, for all M,

$$b'(M) = \frac{P(s', r|s, a, M)b(M)}{\int_{M'} P(s', r|s, a, M')b(M')}, \tag{4.1}$$

where $P(s', r|s, a, M)$ is the probability of transitioning to state s' and observing reward r from state-action pair (s, a) in MDP M. The belief b summarizes the agent's experience in the world. It is a sufficient statistic of history h for predicting future observations, under the usual Bayesian assumption that the prior distribution is correct. For some classes of priors, such as the independent Dirichlet distribution which I use in the experiments, this update can be computed by keeping simple statistics of history such as counts of observations.

During planning, a Bayes-optimal agent considers the effects of its own future changes in belief in addition to changes to the MDP state. I refer to this joint system as the information-state MDP. The full information state is the pair $\langle s, b\rangle$, where s is the MDP state and b is the belief state. Therefore, a state transition in the information-state MDP is from $\langle s, b\rangle$ to $\langle s', b'\rangle$. Next, I define Bayes-optimal planning, which is planning in the information-state MDP.

I define the *mean reward function* given belief b as $R_O(b, s, a) \stackrel{\text{def}}{=} \int_M R_O^M(s, a)b(M)$. The mean reward function is equal to the objective reward function in the information state MDP, because it predicts the immediate expected objective reward when taking action a in the history summarized by belief b and current state a. I emphasize this by overloading the original notation for the objective reward R_O. I similarly define the *mean transition function* as $T(s'|b, s, a) \stackrel{\text{def}}{=} \int_M T^M(s'|s, a)b(M)$.

The optimal behavior in the information state MDP is computed by solving the

Bellman optimality equations: $\forall \langle s, b \rangle \in S \times B, a \in A$,

$$Q^*(\langle s, b \rangle, a) = R_O(b, s, a) + \gamma \sum_{s'} T(s'|b, s, a) V^*(\langle s', b' \rangle),$$

where $V^*(\langle s', b' \rangle) \stackrel{\text{def}}{=} \max_a Q^*(\langle s', b' \rangle, a)$ and b' is the updated belief according to Bayes' rule in equation (4.1) above. The function $Q^*(\langle s, b \rangle, a)$ is the Bayes-optimal action-value function. An agent that acts greedily with respect to $Q^*(\langle s, b \rangle, \cdot)$ acts Bayes-optimally.

Full Bayesian planning (i.e., solving the Bellman optimality equations above for the Bayes-optimal action-value function) is expensive, because for many priors, the set of possible belief states B is large or infinite. Because the agent is constantly learning and updating its belief, an agent may rarely or never (depending on the prior) revisit belief states. Thus, agents must approximate Bayes optimality in general.

4.1.2 Mean MDP with Reward Bonus Planning

Planning with the mean MDP with respect to belief b is a simple, myopic approximation of Bayesian planning which removes the state-space explosion of belief states while preserving physical state dynamics. The resulting Bellman equations are identical to the above equations for Bayesian planning, except the belief state is not updated on the right-hand side:

$$Q_b^*(s, a) = R_O(b, s, a) + \gamma \sum_{s'} T(s'|b, s, a) V_b^*(s'), \tag{4.2}$$

$\forall s \in S, a \in A$, where $V_b^*(s') \stackrel{\text{def}}{=} \max_a Q_b^*(s', a)$. I denote the value's dependence on the current belief b in the subscript to emphasize the belief's invariance during planning. At each time step, the Mean MDP (MMDP) agent acts greedily with respect to Q_b^*. After observing the result of its action, it updates its belief to b' using Bayes' rule and then computes $Q_{b'}^*$. Thus, an agent which plans in the mean MDP *does* update its belief as it receives experience, but does not do so during planning.

I use a reward bonus to help compensate for the information that the mean MDP planning approximation does not account for (cf. Equation 4.2). Such a Mean MDP plus Reward Bonus (MMDP+RB) agent is identical to the mean MDP agent described above, except it uses a reward function defined as the mean reward function plus an added reward bonus term: $\tilde{R}(b, s, a) = R_O(b, s, a) + \hat{R}(b, s, a)$. Formally, an

MMDP+RB agent with belief b always acts greedily with respect to the action-value function defined by: $\forall s \in S, a \in A$,

$$\tilde{Q}_b^*(s,a) = \tilde{R}(b,s,a) + \gamma \sum_{s'} T(s'|b,s,a)\tilde{V}_b^*(s'), \tag{4.3}$$

where $\tilde{V}_b^*(s') \stackrel{\text{def}}{=} \max_a \tilde{Q}_b^*(s',a)$.

The approximate Bayesian algorithm of Kolter and Ng (2009) is an existing algorithm that is of the MMDP+RB form, albeit in a special case. Let $n_{s,a}$ be the number of times that state–action pair (s,a) has been sampled. Kolter and Ng showed that an MMDP+RB agent with a reward bonus of $\hat{R}(b,s,a) = \beta/(n_{s,a} + 1)$ approximates Bayesian planning in polynomial time with high probability (for appropriate choice of constant β), in the special case of an independent Dirichlet prior over transition dynamics per state–action pair and a known reward function. The transition model for the independent Dirichlet distribution given the agent's experience is $T(s'|b,s,a) = \frac{n_{s,a,s'}+\alpha_{s,a,s'}}{\sum_{s'}\left(n_{s,a,s'}+\alpha_{s,a,s'}\right)}$, where the parameters α define the prior and $n_{s,a,s'}$ is the number of times that s' was reached after taking action a in state s. I use this algorithm as one baseline in the experiments.

The MBIE-EB algorithm (Strehl and Littman, 2008) has the same form as MMDP+RB, though it is not derived in a Bayesian setting. In this algorithm, the agent plans using the Maximum-Likelihood Estimate (MLE) of the MDP. Specifically, the model estimate $\hat{T}(s'|s,a) = \frac{n_{s,a,s'}}{\sum_{s'} n_{s,a,s'}}$ from Section 3.2.1 is the MLE transition model for the MDP, which is identical to the mean MDP of an independent Dirichlet prior minus the prior parameters α. MBIE-EB features a reward bonus of $\hat{R}(b,s,a) = \beta/\sqrt{n_{s,a}}$. This algorithm also connects closely to the method proposed here. I expand on this connection in later sections and use the algorithm as another baseline in the experiments.

Both of these baselines share the property that their reward bonuses decrease independently per state–action pair as each is sampled. Both intuitively measure the uncertainty the agent has for that state–action pair. However, neither accounts for information contained in the prior distribution (unless that prior is an independent Dirichlet). In the next subsection, I define the variance-based reward bonus, which is capable of measuring the uncertainty of arbitrary Bayesian priors over environments.

Finally, I note that the variance-based reward bonus in this work is related to the variance reward developed by Duff (2003), though it takes a slightly different form. Duff addressed the problem of designing what he termed an "Optimal Probe." In that work, the agent used *full* Bayesian planning, rather than planning in the Mean

MDP. The goal of Duff's probe was solely to learn the model of the environment as efficiently as possible. The agent had no other objective.

4.1.3 The Variance-Based Reward Bonus

I approximate the level of the agent's uncertainty in the model using the variance in the model parameters. I define the variances of the model parameters as:

$$\sigma^2_{R_O(b,s,a)} \stackrel{\text{def}}{=} \int_M R^M_O(s,a)^2 b(M) - R_O(b,s,a)^2, \text{ and}$$
$$\sigma^2_{T(s'|b,s,a)} \stackrel{\text{def}}{=} \int_M T^M(s'|s,a)^2 b(M) - T(s'|b,s,a)^2.$$

Importantly, these are not the variances of the world dynamics but instead are the variances of the model parameters with respect to the agent's belief—recall that the belief is a distribution over environment models. As the agent gathers experience in the world—as the agent becomes more certain of the truth—this variance decreases to 0, regardless of how stochastic the world is. Notice that I have made no state–action independence requirements on the prior. Therefore, depending on the belief, experience gained in one state may affect the variance term in any other state.

Definition 4.1. Let the variance-based reward bonus be:

$$\hat{R}(b,s,a) \stackrel{\text{def}}{=} \beta_R \sigma_{R_O(b,s,a)} + \beta_T \sqrt{\sum_{s'} \sigma^2_{T(s'|b,s,a)}},$$

for some constants β_R and β_T.

Although the precise form of this reward bonus may seem unintuitive, in the next section, I show that this reward bonus can be used to bound the error of the mean MDP, with respect to the random (drawn from the prior) true MDP with high probability. Using this fact, I show that there exist constants β_R and β_T for which an agent that acts greedily with respect to $\tilde{Q}^*_b(s,a)$ acts optimally with respect to the random true MDP, for all but a polynomially bounded number of time steps with high probability.

4.2 Sample Complexity

The key insight behind the use of variance as a measure of the agent's uncertainty is that one can bound the deviation of the mean MDP from a sample MDP from the posterior with high probability using Chebyshev's inequality. Chebyshev's inequality states that with high probability, the deviation of a random variable from its mean is no more than a multiple of its variance: $\Pr(|X - \mu| \geq \eta) \leq \frac{\sigma^2}{\eta^2}$, where X is a random variable, μ and σ are its mean and standard deviation, and η bounds the deviation from the mean.

I use this inequality to derive the features of the reward function, first for the transition function variance and then for the reward function variance.

Definition 4.2. Define the transition function error bound feature to be:

$$\eta_T(s, a, b) = \frac{1}{\sqrt{\rho}} \sqrt{\sum_{s'} \sigma^2_{T(s'|b,s,a)}}.$$

Lemma 4.1. *For any belief b and any state–action pair (s, a), the value $\eta_T(s, a, b)$ bounds the max-norm error of the mean transition model with probability at least $1 - \rho$:*

$$Pr\left[\left\|T^M(\cdot|s, a) - T(\cdot|b, s, a)\right\|_\infty < \eta_T(s, a, b)\right] > 1 - \rho,$$

where $\|\cdot\|_\infty$ is the max norm.

Proof.

$$\begin{aligned}
&Pr\left[\left\|T^M(\cdot|s, a) - T(\cdot|b, s, a)\right\|_\infty \geq \eta_T(s, a, b)\right] \\
&\leq \sum_{s'} Pr\left[\left|T^M(s'|s, a) - T(s'|b, s, a)\right| \geq \eta_T(s, a, b)\right] \\
&\leq \sum_{s'} \sigma^2_{T(s'|b,s,a)} / \eta^2_T(s, a, b) \leq \rho.
\end{aligned}$$

The first inequality inequality is the result of a union-bound over next-state and the last inequality follows from Definition 4.2. □

Next, I define a similar bound for the reward function. The proof is similar so it is omitted, though it does not require the union bound.

Definition 4.3. Define the reward function error bound feature to be:

$$\eta_R(s, a, b) = \sigma_{R_O(b,s,a)} / \sqrt{\rho}.$$

Algorithm 1: Variance-Based Reward Algorithm

Input: s_0, b_0, β_R, β_T, C
$\forall s, a \quad c_{s,a} \leftarrow 0$, $known_{s,a} \leftarrow false$
$\forall s, a \quad Q(s,a) \leftarrow \tilde{Q}^*_{b_0}(s,a)$
for $t \leftarrow 0, 1, 2, 3, \ldots$ **do**
  $a_t \leftarrow \arg\max_a Q(s_t, a)$
  $r_t, s_{t+1} \leftarrow$ `takeAction`(a_t)
  $b_{t+1} \leftarrow$ `updateBelief`$(b_t, s_t, a_t, r_t, s_{t+1})$
  $c_{s_t,a_t} \leftarrow c_{s_t,a_t} + 1$
  if $\neg known_{s_t,a_t} \wedge (c_{s_t,a_t} \geq C(s_t, a_t, b_0, \epsilon, \delta))$ **then**
    $\forall s, a \; Q(s,a) \leftarrow \tilde{Q}^*_{b_{t+1}}(s,a)$
    $known_{s_t,a_t} \leftarrow true$;
  end
end

Lemma 4.2. *For any belief b and any state–action pair (s,a), if the belief distribution over reward functions has finite variance, then $\eta_R(s,a,b)$ bounds the error of the mean reward function with probability at least $1-\rho$:*

$$Pr\left[\left|R^M_O(s,a) - R_O(b,s,a)\right| < \eta_R(s,a,b)\right] > 1 - \rho.$$

Although the variance reward MMDP+RB agent was described in detail in Section 4.1.2, I analyze a slightly different agent in the theoretical analysis, presented in Algorithm 1. It differs in only one significant manner. Algorithm 1 takes as input a sample complexity parameter $C(s,a)$, which is a prior-dependent term indicating the number of times state–action pair (s,a) must be sampled before it becomes *known*. Algorithm 1 only updates its value function estimate each time a state–action pair becomes known. This allows me to bound the number of times the agent plans.

In this section, I bound the sample complexity of Algorithm 1, following the abstract PAC framework by Strehl et al. (2006b). A central aspect of this framework is the principle of optimism in the face of uncertainty.

An agent is optimistic if its value estimates are greater than the true value the MDP it is estimating. Specifically, let $V^*_M(s)$ denote the value function for MDP M. It is given by the solution to the Bellman equation:

$$V^*_M(s) = \max_a R^M_O(s,a) + \gamma \sum_{s'} T^M(s'|s,a) V^*_M(s').$$

An MMDP+RB agent is optimistic with respect to MDP M if $\tilde{V}^*_b(s) > V^*_M(s) \, \forall s$.

Optimism ensures that the agent will not ignore potentially lucrative opportunities. This results in exploration, because the agent will take actions which are uncertain but can potentially yield large expected objective return.

Lemmas 4.1 and 4.2 above allow me to provide a reward bonus that ensures optimism with high probability. Let the variance-based reward bonus be

$$\hat{R}(b,s,a) = \frac{1}{\sqrt{\rho}} \left(\sigma_{R_b(s,a)} + \frac{\gamma S}{1-\gamma} \sqrt{\sum_{s'} \sigma^2_{T(s'|b,s,a)}} \right). \tag{4.4}$$

Thus let, $\beta_R = \frac{1}{\sqrt{\rho}}$ and $\beta_T = \frac{\gamma S}{\sqrt{\rho}(1-\gamma)}$.

Notice that the variance-based reward bonus in fact uses the square-root of the variance. Thus, it could more accurately be called the standard-deviation-based reward bonus. I use the term "variance-based" for conciseness.

Next, I state the formal results about the use of this reward bonus in an MMDP+RB agent. Unless stated otherwise, the proofs in this section are deferred to Section 4.5 at the end of this chapter.

Lemma 4.3 (Optimism)**.** *Let the reward bonus be as defined in Equation (4.4), then the value function computed by Algorithm 1 is optimistic with probability at least* $1 - 2S^2A^2\rho$ *for every planning step during its execution.*

The learning rate of Algorithm 1 depends on the convergence rate of the posterior distribution. I abstractly define this rate as a function f of the initial belief distribution and other parameters (f is similar to but slightly different from the function f in Asmuth et al. 2009). I later provide examples of this function for specific classes of priors. Note that because I have defined the reward bonus in such a way that it is an upper bound on the error of the mean MDP, defining the sample complexity with respect to the reward bonus term bounds the number of samples before I have an accurate model.

Definition 4.4. Define the sample complexity function, $f(b_0, s, a, \epsilon, \delta, \rho)$, as the minimum number c such that $\forall d > c$, if d transitions from (s,a) have been observed starting from belief b_0, the reward bonus $\hat{R}$ with the updated belief b_d is less than ϵ, i.e., $\hat{R}(b_d, s, a) = \frac{1}{\sqrt{\rho}} \left(\sigma_{R_{b_d}(s,a)} + \frac{\gamma S}{1-\gamma} \sqrt{\sum_{s'} \sigma^2_{T(s'|b_d,s,a)}} \right) < \epsilon$, with probability at least $1 - \delta$. I will refer to a state–action pair as *known* if it has been sampled c or more times.

I make one important assumption about the sample complexity function f: experience gained from sampling state–action pair (s,a) does not increase the sample

complexity of another state–action pair (s', a'). This assumption is trivially true when the model parameters of different state–action pairs are independent of each other in the prior. Furthermore, I expect this to be a reasonable assumption in many correlated priors. In fact, I believe the performance of the variance-based reward bonus on correlated priors is one of its strengths. Rather than hurting convergence, I expect experience gained from one area of the state space to reduce the number of samples required from another. The intuition here is that information gathered in other states shouldn't hurt the agent, and can often help. I present evidence consistent with this below.

Next, I present this chapter's central theoretical result that bounds the number of time steps it takes to achieve optimal performance with respect to the true MDP. This result is distinct from the sample complexity result of Kolter and Ng (2009) which bounds the time it takes to act optimally with respect to the information state MDP.

Theorem 4.1. *Let the sample complexity of state s and action a be $C(s,a) = f(b_0, s, a, \frac{1}{4}\epsilon(1-\gamma)^2, \frac{\delta}{SA}, \frac{\delta}{2S^2A^2})$. Let the reward $\hat{R}$ be defined as in Lemma 4.3 with $\rho = \frac{\delta}{2S^2A^2}$. Let M be the random true model MDP distributed according to the prior belief b_0. Algorithm 1 will follow a 4ϵ-optimal policy from its current state, with respect to the true MDP M, on all but*

$$O\left(\frac{\sum_{s,a} C(s,a)}{\epsilon(1-\gamma)^2} \ln \frac{1}{\delta} \ln \frac{1}{\epsilon(1-\gamma)}\right) \tag{4.5}$$

time steps with probability at least $1 - 4\delta$.

4.2.1 The Variance-Based Reward Bonus Extends MBIE

Theorem 4.1 can be applied to many prior distributions. In the remainder of this section, I apply it to two simple special cases. First, I provide a concrete bound in the case of an independent Dirichlet prior and a known reward function. I use this special case to connect Theorem 4.1 to related work.

Lemma 4.4. ***(Independent Dirichlet Prior)*** *Let $n_{s,a}$ be the number of times state–action pair (s,a) has been sampled. For a known reward function and an independent Dirichlet prior over next-state transition dynamics for each state–action pair, the reward feature $\eta_T(s,a,b)$ decreases at a rate of $O(1/\sqrt{n_{s,a}})$.*

Proof. The square root of the sum of the variance terms for the Dirichlet distribution

is

$$\sqrt{\sum_{s'} \sigma^2_{T(s'|b,s,a)}} = \sqrt{\frac{\sum_{s'} T(s'|b,s,a)(1 - T(s'|b,s,a))}{n_{s,a} + 1}}$$
$$\leq 1/\sqrt{n_{s,a} + 1}.$$ □

Lemma 4.5 uses the result in Lemma 4.4 to provide a sample complexity bound for Algorithm 1 with a Dirichlet prior.

Lemma 4.5. *For an independent Dirichlet prior over transition dynamics and a known reward function* $f(b_0, s, a, \epsilon, \delta, \rho) = O(\gamma^2 S^2 / (\rho \epsilon^2 (1-\gamma)^2))$*, where* S *is the number of states.*

Proof. According to the definition of f, the sample complexity is the number of samples required before the reward bonus drops below ϵ. Because I am considering the case of a known reward function, this translates to:

$$\hat{R}(b, s, a) = \frac{\gamma S}{\sqrt{\rho}(1-\gamma)} \sqrt{\sum_{s'} \sigma^2_{T_b(s'|s,a)}} < \frac{\gamma S}{\sqrt{\rho}(1-\gamma)} \frac{1}{\sqrt{n_{s,a} + 1}} < \epsilon$$

$$n_{s,a} + 1 > \frac{\gamma^2 S^2}{\rho \epsilon^2 (1-\gamma)^2}.$$ □

As stated before, the mean MDP for a Dirichlet prior is analogous to the MLE estimate MDP in MBIE-EB. Notice also that the $O(1/\sqrt{n_{s,a}})$ reward bonus derived here is similar to the reward bonus in MBIE-EB. In other words, I have effectively re-derived MBIE-EB using Bayesian methods; one could replace the variance term with a $1/\sqrt{n_{s,a} + 1}$ reward bonus and produce a similar result.

This connection has important implications for the claim in the previous chapter that reward design is for the purpose of overcoming agent limitations. The MBIE-EB algorithm was originally derived without viewing the original planning agent as an approximation of a more-expensive optimal learning agent. However, the results here indicate that the reward bonus in MBIE-EB can be interpreted as mitigating the limitation arising from using the mean MDP approximation in a Bayesian agent with an independent Dirichlet prior.

4.2.2 Small Sample Complexity for a Prior over Deterministic Environments

The advance of Algorithm 1 over previous approaches lies in its ability to take advantage of structured and informative priors. In this section, I show by example that strong sample complexity bounds can be achieved given an informative prior. Specifically, I present the sample complexity function f for a prior over unknown deterministic MDPs. This prior is more informative than the Dirichlet prior, which has positive probability for both deterministic and stochastic MDPs.

Lemma 4.6. ***(Prior over Deterministic MDPs)*** *Let b_0 be a prior over deterministic worlds. The sample complexity function f is constant: $f(b_0, s, a, \epsilon, \delta, \rho) \leq 1$.*

Proof. By Bayes' rule, the posterior probability of any model which has a transition which is inconsistent with observed data will be 0 after one sample. Therefore, the variance terms corresponding to an experienced state–action pair will be 0 after one sample. Once a model has 0 probability under the posterior, it will always have 0 probability after a Bayes' rule update. □

4.3 Empirical Results

The proposed agent benefits from the Bayesian prior over MDPs in two ways: (1) it uses the prior to generate the Mean MDP; (2) it uses the variance calculation to guide exploration. To properly demonstrate that the variance reward bonus deserves the credit for the agent's success, all comparison methods will be given the same prior belief, and all will properly update their posterior belief given the prior.

The simplest baseline for comparison is the mean MDP agent with no reward bonus. To fairly compare against MBIE-EB (Strehl and Littman, 2008) and the approximate Bayesian method of Kolter and Ng (2009), I test the corresponding MMDP+RB agents with reward bonuses of $O(1/\sqrt{n_{s,a}})$ and $O(1/n_{s,a})$, respectively. For both of these reward bonuses, and for the variance-based reward bonus, I solve the corresponding optimal reward problem (where the parameters β are the degree to which to scale the reward bonuses). I then compare the performances of the optimal reward functions from each search space. This is the first instance in this thesis of different reward function search spaces being compared on the same agent. This process can also be viewed as solving one big optimal reward problem, where the overall optimal reward

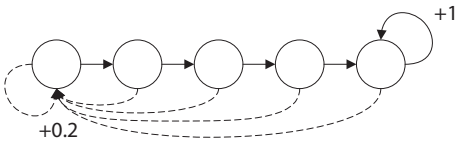


Figure 4.1: Chain Environment

function is the reward function which is optimal across all the reward sets. Breaking the ORP up in this way allows for comparison to related work.

The BOSS algorithm (Asmuth et al., 2009) is *not* a reward-design approach, but it is a direct competitor to the variance-based reward method in another sense. It is the only other algorithm I am aware of which provides sample complexity guarantees as a function of arbitrary Bayesian priors. Each time it plans, it samples K (a parameter) MDPs from the posterior distribution. It then plans in a combined MDP that has the same state space, but each state has $K \times A$ available actions. Essentially, the combined MDP allows the agent to choose, independently in each state, which sampled MDP's dynamics it would like to follow. This planning method is optimistic with enough samples K.

There are many other approximate methods for Bayesian planning. By using a standard benchmark task below, I am able to compare against the published results for one such method. The BEETLE algorithm (Poupart et al., 2006) directly approximates full Bayesian planning by compressing the information state space.

I first compare the methods on a standard benchmark problem, and then on a problem with an interesting structured prior.

4.3.1 Chain Environment

The 5-state chain environment shown in Figure 4.1 is a common benchmark environment used in related work on Bayesian exploration. In this section, I use it to compare the variance-based reward bonus method to related work. The chain environment has two actions: Action A (solid) advances the agent along the chain, and Action B (dashed) resets the agent to the first node. When taken from the last node, Action A leaves the agent where it is and gives the designer an objective reward of 1—otherwise it results in 0 objective reward. Action B gives a reward of 0.2 in all states. However, for both actions, with probability 0.2 the agent "slips" and the next-state distribution and reward of the other action results instead. Optimal behavior always chooses

Algorithm	Tied Prior	Semi Prior	Independent Prior
BEETLE	3,650	3,648	1,754
BOSS	3,657	3,651	3,003
Mean	3,642	3,257	3,078
$O(1/n)$	3,645	3,642	3,430
$O(1/\sqrt{n})$	3,645	3,642	3,462
Variance Reward	3,645	3,637	3,465

Table 4.1: Chain Experiment Results. Each value is the cumulative reward obtained over 1000 steps, averaged over 500 trials.

Action A.

This environment was designed to require smart exploration, because the optimal policy produces distant reward while there are many sub-optimal policies which yield immediate reward. Past work (Poupart et al., 2006; Asmuth et al., 2009) considers the performance of agents with different priors in this environment. In the *Independent* prior, the agent uses an independent Dirichlet prior distribution for each state–action pair. Under the *Tied* prior, the agent knows the underlying transition dynamics except for the value of a single slip probability that is shared between all state–action pairs. The *Semi*-tied prior allows for a different slip probability for each action which is shared across states. In Tied and Semi, the prior distribution over the slip probability is represented as a Beta distribution. In keeping with published results on this problem, Table 4.1 reports cumulative return in the first 1000 steps, averaged over 500 trials. Standard error is on the order of 20 to 50. The optimal policy in the chain environment, after the dynamics are learned, achieves an expected return of 3677.

I present the performance of the variance-based reward agent alongside the comparison methods in Table 4.1. Each comparison algorithm is given the indicated prior and maintains the correct posterior distribution; however, each differs in its method used to approximate full Bayesian planning.

For each of the reward-based methods, I solved the resulting optimal reward problem using the same adaptive sampling procedure used in Chapter 3. In Table 4.1, the ORP was solved separately for each entry.

The reward bonus methods, including the variance-based reward, perform as well as the other methods in general and outperform the alternatives in the case of the independent prior. As predicted, the reward bonus methods perform similarly in the case of the independent (Dirichlet) prior. Although the Tied and Semi priors are structured, they essentially make the problem too easy—other than from the naive mean MDP approach, they fail to differentiate the methods.

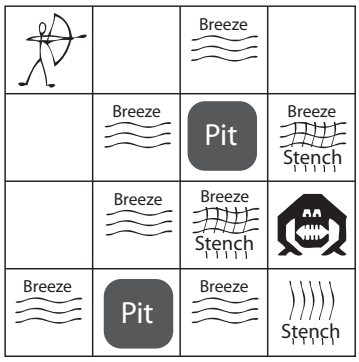


Figure 4.2: Hunt the Wumpus Environment

4.3.2 Hunt the Wumpus

Next, I demonstrate the advantages of the variance-based reward on a task with non-trivial correlated belief reasoning requirements, and in which poor early decisions can lead to the agent's death. The Hunt the Wumpus environment, adapted from Russell and Norvig (2002) and pictured in Figure 4.2, is a discrete world based on an old computer game which requires intelligent exploration. The world consists of a 4 $\times$ 4 cave. The agent always starts in the top-left corner. It can navigate by turning `left`, turning `right`, or moving `forward` one location. Lurking in the cave in a uniformly random location (other than the agent's starting location) is the wumpus, a beast that eats anyone who enters its location (ending the episode). The agent cannot see the wumpus, but if it is in a location adjacent to the wumpus in a cardinal direction, it can smell a vile stench. Each location also has a prior 0.2 probability of containing a deep pit that will trap a wandering adventurer (but not the wumpus), ending the episode. If the agent is next to any pit, it can feel a breeze, though it cannot sense in what direction the originating pit is. The agent carries a bow with one arrow and has a `shoot` action. When executed, it fires an arrow the entire length of the cave in the direction it is facing. If it hits the wumpus, the wumpus dies and the designer receives 1 objective reward. Otherwise, the episode ends with 0 reward. At all other time steps, the designer receives a small penalty of -0.01 objective reward.

The agents are evaluated in a setting in which they have *one chance* to kill the wumpus. Specifically, the pit and wumpus locations are resampled from the prior between episodes. Because the world's dynamics can be expressed as a function of the agent's observable location, and because they do not change during an episode, the each episode can be modeled as a Bayesian distribution over MDPs. I define each

	Parameter	Objective Reward/Episode
Variance	$\beta_T = 0.24$	0.508
$O(1/n)$	$\beta = 0.012$	0.293
$O(1/\sqrt{n})$	$\beta = 0.012$	0.291
BOSS	$K = 20$	0.183
Mean MDP	N/A	0.266

Table 4.2: Hunt the Wumpus Results.

MDP as follows: the agent senses the tuple $\langle location, orientation, stench, breeze \rangle$. Each tuple corresponds to different state. The end conditions are represented with two terminal states. In one, the designer receives 1 reward for the agent killing the wumpus. In the other, the designer receives 0 for the agent dying. The dynamics between states are defined above I provide the agent with the true prior over MDPs as defined above at the beginning of each episode.

However, this environment is an unforgiving exploration benchmark; if the agent over-explores, it can fall into a pit or get eaten. There is no opportunity to learn from dying, because the agent only lives for one episode. In order to avoid likely death, while taking enough risks to find and kill the wumpus, an agent will need to properly utilize its prior.

In spite of the fact that the dynamics are deterministic, probabilistic Bayesian reasoning is necessary due to the stochastic nature of the prior. It is often the case that, given the agent's experience, some adjacent locations are more likely to contain pits than others. Information gathered in one location can increase or decrease the probabilities of the existence of pits or the wumpus in other locations. At times, the agent must take calculated risks—stepping into locations that have nonzero probability of containing a pit—in order to gain the information it needs to locate the wumpus.

For example, Russell and Norvig (2002, Section 13.7) examine a situation in which, based on its prior experience, the agent cannot yet locate the wumpus, and among the three unvisited locations adjacent to the positions it has visited, two potential locations to explore have a 0.31 chance of containing a pit while one of them has a 0.86 chance of containing a pit. For the variance-based reward agent, the objective reward accounts for the chance of death, motivating the agent to avoid these locations proportional to the chance of dying, while the variance-based reward bonus motivates the agent to explore these locations to gain information about the environment. By adding these reward signals together, the agent balances these opposing goals and takes an action which leads it to one of the safer exploratory locations.

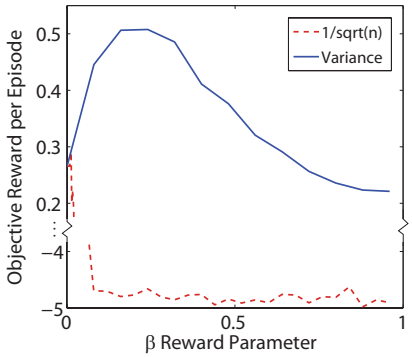


Figure 4.3: Reward Sensitivity in Hunt the Wumpus

I present the empirical results in Table 4.2. For each method, I present the mean objective reward obtained per episode, averaged over 500 episodes. Each episode is capped at a maximum length of 1,000 steps. As before, each reported method is given the true prior and properly updates its belief through experience; the algorithms differ in how they approximate Bayesian planning given those beliefs. For the optimal reward problems, I search over all reward bonus scalars β in the range $[0, 0.04]$ in increments of 0.002 and $[0.04, 1]$ in increments of 0.04. For the variance method, I present β_T only, because the reward function is known (and the corresponding variance term is 0). For the BOSS method, I tested sample sizes (K) of 1,5,10,20,40, and 80.

The variance reward method achieves the top performance, because it follows an effective controlled exploration strategy. It is not optimal, however; because it enjoys exploring, it will occasionally spend time identifying the location of a few more pits after having already located the Wumpus. The BOSS agent performs poorly. As mentioned above, BOSS ensures optimism by building in the assumption that the agent knows, and in fact is in control of, which MDP the agent is in. Its control policy immediately turns and fires an arrow at time step 1—it chooses the imagined MDP in which the Wumpus is in its current line of sight. The mean MDP baseline agent performs better than BOSS, though its policy is merely a slightly better heuristic. It walks in a straight line until it encounters a breeze, at which point it fires in the direction of the most unexplored locations, unless it experiences a rare situation which disambiguates the wumpus's location, in which case it fires in the correct direction. The $O(1/n)$ and $O(1/\sqrt{n})$ reward bonus methods' policies are a small deviation from the mean MDP's policy—notice the small magnitude of the reward coefficient, just above the per-step penalty of the objective reward.

Figure 4.3 illustrates the effect of the choice of the reward scaling parameter β on

performance for both the variance and $O(1/\sqrt{n})$ reward methods (the graph is similar for $O(1/n)$). Note that the choice of $\beta = 0$ results in the mean MDP agent. As can be seen, use of the variance-based reward function results in good performance for many choices of β, but $O(1/\sqrt{n})$ performs extremely poorly for large reward values—the agent spends the majority of its time following safe actions, such as turning in circles, that provide it with no information.

4.4 Discussion

Although full Bayesian planning produces optimal behavior, it is intractable. In this work, I contributed a novel reward bonus that approximates Bayes-optimal learning and produces provably efficient exploration with high probability. The method is similar to existing approaches when given unstructured priors, such as the independent Dirichlet distribution; however, unlike previous reward bonuses, the variance-based reward bonus approach is capable of exploiting an informative prior. In addition to providing theoretical results supporting these claims, I demonstrated that the variance-based reward bonus exploits a prior with correlated knowledge between states in the Hunt the Wumpus environment.

The reward bonuses discussed in this chapter—variance, $1/\sqrt{n}$, and $1/n$—can all be used as features in an optimal reward problem, as was done above in the empirical section. The theorems involved in these features' derivations, such as Theorem 4.1, or the corresponding theorems in Strehl and Littman (2008) or Kolter and Ng (2009), each prescribe a specific setting of β in order for the theorem to hold for a desired ϵ and δ. In practice, however, the optimal β tends to be much smaller than the one prescribed by the theorem. In general, I expect these will be part of a larger optimal reward problem with many reward features. Experiments 3 and 4 in Chapter 3, for example, use the $1/n$ in conjunction with the *local-model-inaccuracy* feature.

In order to make progress towards this broader objective, one needs to be able to solve an optimal reward problem with many features. The brute-force offline-algorithms I have been using so far in this dissertation do not scale. In the next chapter, I present a gradient-ascent algorithm, which is not only able to solve an ORP with thousands of features, it is able to approximately solve an ORP online within a single agent's lifetime.

4.5 Proofs

Proof of Lemma 4.3. Define the random variable $V^*_M(s)$ to be the value of state s given the true model M. Let $\eta_T(s,a,b)$ and $\eta_R(s,a,b)$ be defined as in Lemmas 4.1 and 4.2. Given belief b, with probability at least $1 - 2SA\rho$,

$$\begin{aligned} V^*_M(s) &= \max_a R^M_O(s,a) + \gamma \sum_{s'} T^M(s'|s,a) V^*_M(s') \\ &\leq \max_a R_O(b,s,a) + \eta_R(s,a,b) \\ &\qquad + \gamma \sum_{s'} (T(s'|b,s,a) + \eta_T(s,a,b)) V^*_M(s') \\ &\leq \max_a \tilde{R}(b,s,a) + \gamma \sum_{s'} T(s'|b,s,a) \tilde{V}^*_b(s') = \tilde{V}^*_b(s). \end{aligned}$$

The first inequality is true with probability at least $1 - 2\rho$, and the final step can be shown by induction. This must be true for all state–action pairs, which holds with probability greater than $1 - 2SA\rho$ by a union bound. For this to be true for the entire execution of Algorithm 1, it must be true for all value function updates, of which there are no more[2] than SA, resulting in the final bound of $1 - 2S^2A^2\rho$ through another application of the union bound. □

Definition 4.5. Let $M = \langle S, A, T, R, \gamma \rangle$ be an MDP. Given a set of Q value estimates $Q(s,a)$ and a set K of state–action pairs called the *known* state–action pairs, I define the *known state–action MDP* $M_K = \langle S \cup s_0, A, T_K, R_K, \gamma \rangle$ as follows. For each known state–action pair $(s,a) \in K$, $T_K(\cdot|s,a) = T(\cdot|s,a)$ and $R_K(s,a) = R(s,a)$. Define an additional absorbing state s_0 with $T_K(s_0|s_0,\cdot) = 1$ and $R_K(s_0,\cdot) = 0$. For each unknown state–action pair, the world deterministically transitions to s_0 $(\forall (s,a) \notin K, T(s_0|s,a) = 1)$ with reward function $R_K(s,a) = Q(s,a)$.

Let $V_t(s) = \max_a Q_t(s,a)$ denote the agent's estimate of the value function at time t. Let $V^{\pi_t}_{M_{K_t}}$ denote the value of the agent's policy at time t in the known state–action MDP defined with respect to the true MDP M and the agent's value estimate Q_t.

Lemma 4.7. *If the state–action sample complexity in Algorithm 1 is* $C(s,a) = f(b_0, s, a, \frac{1}{4}\epsilon(1-\gamma)^2, \frac{\delta}{SA}, \frac{\rho}{2S^2A^2})$ *then* $V_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq \epsilon$ *for all time steps* t *with probability at least* $1 - \delta - \rho$.

Proof. Let $\delta' = \frac{\delta}{SA}$. Because the sample complexity bound holds for each state–action

[2]Technically, including the initial planning phase, there are up to $SA + 1$ value updates.

pair with probability δ', the sample complexity bound holds overall by union bound with probability $1 - SA\delta' = 1 - \delta$.

For the remainder of the lemma, it suffices to show that the mean MDP has low error in known states. Let $A = \frac{1}{4}\epsilon(1-\gamma)^2$. In the known states, $\hat{R}(b,s,a) < A$; therefore $\eta_R(s,a,b) < A$ and $\eta_T(s,a,b) < \frac{(1-\gamma)A}{\gamma S}$. These imply $\left|R_O^M(s,a) - R_O(b,s,a)\right| < A$ and $\left\|T^M(\cdot|s,a) - T(\cdot|b,s,a)\right\|_\infty < \frac{(1-\gamma)A}{\gamma S}$. For the transition bound, I can convert the max norm bound into an L^1 norm by multiplying by S: $\left\|T^M(\cdot|s,a) - T(\cdot|b,s,a)\right\|_1 < \frac{(1-\gamma)A}{\gamma}$. It can be shown by Lemma 1 in Strehl and Littman (2008) that these bounds on the reward function and transition function errors lead to at most $|V_t(s) - V_{M_{K_t}}^{\pi_t}(s)| < \frac{2A}{(1-\gamma)^2} = \frac{\epsilon}{2}$ error, if using the mean reward function in known states. Because Algorithm 1 uses the reward bonus even in known states, there is an additional error less than $\frac{A}{1-\gamma}$. However, $\frac{A}{1-\gamma} + \frac{\epsilon}{2} < \epsilon$. □

Proof of Theorem. This theorem is a straightforward application of Proposition 1 in Strehl et al. (2006b). The theorem requires three conditions: (1) Optimism: $V_t(s) \geq V^*(s) - \epsilon$. This was shown in Lemma 4.3 to hold with probability $1 - \delta$. (2) Accuracy: $V_t(s) - V_{M_{K_t}}^{\pi_t}(s) \leq \epsilon$. This was shown in Lemma 4.7 to hold with probability $1 - 2\delta$. (3) Sample complexity: The number of escape events is bounded by $\sum_{s,a} C(s,a)$. These three conditions are sufficient to prove the stated bound. □

Chapter 5

Reward Design via Online Gradient Ascent[1]

The previous chapters have focused on properties of optimal reward functions and their correspondence to the agent architecture and the environment. Thus, I have used inefficient exhaustive search methods for finding good approximations to the optimal reward function R^*. Given a discrete set of reward functions $\mathcal{R}$, the agent was repeatedly re-evaluated using each reward, and the best-performing reward was selected as the approximately optimal one. There are several disadvantages with this approach. Exhaustively searching an enumerated set does not scale well as the number of reward function features increases. It doesn't take advantage of knowledge gained within an agent's lifetime and doesn't take advantage of knowledge the agent designer has about the agent architecture. As discussed in Chapter 2, there are some existing approaches which attempt to overcome some of these difficulties.

Many approaches involve learning good potential-based shaping reward functions (Ng et al., 1999; Konidaris and Barto, 2006; Marthi, 2007; Grześ and Kudenko, 2008, 2009). Some of these approaches learn the reward functions within an agent's lifetime. However, these approaches do not solve the optimal reward problem. Instead, they generally attempt to learn the potential-based reward function which corresponds to the optimal value function V^*. In the next chapter (Chapter 6), I demonstrate that potential-based reward functions, and the V^* potential-based reward in particular, can be suboptimal.

One approach is to use genetic programming methods to optimize reward functions (Uchibe and Doya, 2008; Elfwing et al., 2008; Meriçli et al., 2010; Niekum et al., 2010). These methods propose a solution to the scaling problem above, in that genetic programming is a more efficient method of searching the reward space than brute-force search. However, they still suffer from the problem that each reward function is

[1]This chapter presents material from Sorg et al. (2010b).

evaluated independently in a separate agent lifetime to obtain estimates of its quality, and that these methods are generic optimization methods that ignore properties of the agent.

In this chapter I present Policy Gradient for Reward Design (PGRD), an online stochastic gradient ascent method for approximately solving the optimal reward problem. I show that this algorithm is capable of improving reward functions in agents with a variety of limitations, including bounds on the planning horizon and the use of an inaccurate model. PGRD has few parameters and has a linear (both time and memory) per-time-step complexity in the number of reward function parameters. Unlike previous approaches, it takes advantage of knowledge about the agent's structure (through the gradient computation). To my knowledge, this is the first algorithm designed to approximately solve the optimal reward problem in an online setting—during a single agent's lifetime.

In Section 5.1, I present the PGRD algorithm, prove its convergence, and show that it is a generalization of the policy gradient method OLPOMDP (Bartlett and Baxter, 2000), upon which it is based. In Section 5.2, I present experiments demonstrating PGRD's ability to approximately solve the optimal reward problem online.

5.1 PGRD: Policy Gradient for Reward Design

As in previous chapters, I assume the reward function space is parameterized by a vector of parameters β chosen from space of parameters $\mathbb{R}^m$. Each $\beta \in \mathbb{R}^m$ specifies a reward function $R(\cdot;\beta)$. Furthermore, in this chapter, I assume the reward function is differentiable with respect the parameters β. The precise form of the reward function is defined below.

PGRD builds on the following insight: a planning agent procedurally converts the reward function into behavior; thus, the reward function parameters can be viewed as parameters of the agent's policy. Using this insight, PGRD optimizes the reward function parameters by estimating and ascending the gradient of the objective return with respect to the reward parameters, $\nabla_\beta \mathbb{E}\left[U_{R_O}(h)|h \sim M\langle G(R(\cdot;\beta))\rangle\right]$, from experience, using standard policy gradient techniques. In fact, I show that PGRD can be viewed as an (independently interesting) generalization of the policy gradient method OLPOMDP (Bartlett and Baxter, 2000). Specifically, I show that OLPOMDP is special case of PGRD when the agent's planning horizon H is one[2]. In this section, I

[2] In Sorg et al. (2010b), the planning algorithm is defined differently, such that PGRD is equivalent

first present the family of local planning agents for which PGRD improves the reward function. Next, I develop PGRD and prove its convergence. Finally, I show that PGRD generalizes OLPOMDP and discuss how adding planning to OLPOMDP affects the space of policies available to the optimization method.

Assumptions on Environment and Objective Return In this chapter, I assume that the environment can be modeled as a Partially-Observable Markov Decision Process (POMDP) with a finite number of hidden-states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, and observations $o \in O$. The dynamics are governed by a state-transition function $P(s'|s,a)$ that defines a distribution over next-states s' conditioned on current state s and action a, and an observation function $\Omega(o|s)$ that defines a distribution over observations o conditioned on current state s. The environment structure of the hidden states and the assumption of finite sets are useful for the theorems in this chapter, but the algorithm can handle environments without this structure and with infinite sets in practice.

In this chapter, I assume that the objective return is the expected mean objective reward obtained over an infinite horizon, i.e., $U_{R_O}(h) = \lim_{N\to\infty} \mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N} R_O(o_t, a_t)\right]$.

A Family of Limited Planning Agents

PGRD optimizes the reward function for a class of agents that performs repeated local planning from its current state to select its next action. Here, I define this class of agents.

Model I assume the agent G is given a fixed Markov model T defined over the observation space O and action space A, denote $T(o'|o,a)$ the probability of next observation o' given that the agent takes action a after observing o. In general, PGRD can improve reward functions in agents which use non-Markov models defined in terms of the history of observations and actions; this is a straightforward extension of the approach here. The agent uses its model T to plan at each step and acts according to this plan. I do not assume that the model T is an accurate model of the environment. The use of an incorrect model is one type of agent limitation I examine in the experiments.

to OLPOMDP at horizon $H = 0$. In this dissertation, I define the planning algorithm's horizon to be consistent with other chapters.

Internal State. The agent G maintains an internal state i from an assumed finite set of internal states I. Each time step, the agent's internal state is updated at each time step using the function $i_{t+1} = \texttt{updateInternalState}(i_t, a_t, o_{t+1})$. The assumption of a finite internal state space is useful in the central theorem proof of this chapter, but PGRD is readily applied on agents with an uncountable number of internal states.

The agent chooses actions according to a stochastic policy $\mu(a|i;\beta)$, which is a distribution over $a \in \mathcal{A}$ given internal state i, parameterized by the reward parameters β. Thus, given fixed reward parameters β, the internal state fully describes the information used by the agent when making its decision. The internal state is the agent's compact representation of its history h. This can encompass any type of knowledge that is updated with experience and is used by the agent in decision making. In this chapter, the internal state allows the agent to use reward functions that depend on the agent's history. The function `updateInternalState` is used to update statistics of history used by the reward function.

Reward Function. I consider reward functions of the form $R(i, o, a; \beta) = \beta^\mathsf{T}\phi(i, o, a)$, where β is the reward parameter vector, and $\phi(i, o, a)$ is a vector of features based on internal state i, planning state o, and action a. Note that if ϕ is a vector of binary indicator features, this representation allows for arbitrary reward functions and thus the representation is completely general. The internal state is capable of capturing summary statistics of history that have been used in reward functions in previous chapters. The reward functions in Chapter 3, which recorded sample counts and inverse-recency counts, have simple `updateInternalState` functions, though they do not satisfy the assumption that the internal state space is finite. The local-model-inaccuracy feature and the variance-based reward feature are other examples which have more complicated `updateInternalState` functions.

Planning Algorithm. At each time step t, the agent's planning algorithm, `plan`, performs Full Finite-Horizon Planning (FFHP)[3]. Specifically, the agent computes the optimal policy on the finite-horizon problem rooted at the current (assumed Markov) observation o_t using its model T and reward function $R(i_t, o, a; \beta_t)$ with current internal state i_t and reward parameters β_t. Specifically, the agent computes

[3]PGRD is well-defined and converges for infinite-horizon planning ($H = \infty$) also.

an H-step-optimal Q-value function $Q^H(i_t, o_t, a; \beta_t)\ \forall a \in \mathcal{A}$, where

$$Q^H(i_t, o, a; \beta_t) = R(i_t, o, a; \beta_t) + \gamma \sum_{o' \in O} T(o'|o, a) \max_{b \in \mathcal{A}} Q^{H-1}(i_t, o', b; \beta_t) \tag{5.1}$$

and $Q^1(i_t, o, a; \beta_t) = R(i_t, o, a; \beta_t)$. I emphasize that the internal state i_t and reward parameters β_t are held invariant while planning. Notice that this is identical to the FFHP definition in Chapter 3 (Equation 3.1), but history is being summarized by internal state. Note that the H-step Q-values are computed only for the current observation o_t, in effect by building a horizon-H tree rooted at o_t. In the $H = 1$ special case, the planning procedure completely ignores the model T and returns $Q^1(i_t, o_t, a; \beta_t) = R(i_t, o_t, a; \beta_t)$. Regardless of the value of H, I treat the end result of planning as providing an *action scoring function* $Q_t(a; \beta_t)$ where the dependence on H, i_t and o_t is dropped from the notation. To allow for gradient calculations, the agents act according to the Boltzmann (softmax) stochastic policy parameterized by Q:

$$\mu(a|i_t; Q_t) \stackrel{\text{def}}{=} \frac{e^{\tau Q_t(a;\beta_t)}}{\sum_b e^{\tau Q_t(b;\beta_t)}},$$

where τ is a temperature parameter that determines how stochastically the agent selects the action with the highest score. Note that μ is still fundamentally parameterized by β, because Q is parameterized by β. When the planning horizon H is small due to computational limitations, the agent cannot account for events beyond the planning horizon. I examine this limitation in the experiments.

The advantage gained by incorporating model-based planning into a policy gradient algorithm is that it allows exploitation of the knowledge contained in the model. The concomitant potential disadvantage is that if the model is wrong, exploiting it can be detrimental. However, I demonstrate that even if the model is wrong, PGRD can be beneficial.

5.1.1 Gradient Ascent of Objective Return

In order to improve the agent's reward function via gradient ascent, PGRD needs to compute the gradient of the objective return with respect to the agent G's reward function parameters β, $\nabla_\beta \mathbb{E}\left[U_{R_O}(h)|h \sim M\langle G(R(\cdot;\beta))\rangle\right]$. The main insight is to break the gradient calculation into the calculation of two gradients. The first is the gradient of the objective return with respect to the agent's policy μ, and the second is the

gradient of the policy with respect to the reward function parameters β. The first gradient is exactly what is computed in standard policy gradient approaches (Bartlett and Baxter, 2000). The second gradient is challenging because the transformation from reward parameters to policy involves a model-based planning procedure. I draw from the work of Neu and Szepesvári (2007) which shows that this gradient computation resembles planning itself. I develop PGRD, presented in Algorithm 2, explicitly as a generalization of OLPOMDP, a policy gradient algorithm developed by Bartlett and Baxter (2000), because of its foundational simplicity relative to other policy-gradient algorithms such as those based on actor-critic methods (e.g., Bhatnagar et al., 2009). In principle, any policy-gradient approach would apply. Notably, the reward parameters are the only parameters being learned in PGRD.

5.1.2 The Gradient of the Policy with respect to Reward

For the Boltzmann (softmax) distribution, the gradient of the policy with respect to the reward parameters is given by

$$\nabla_{\beta_t}\mu(a|i_t;Q_t) = \tau\cdot\mu(a|i_t;Q_t)[\nabla_{\beta_t}Q_t(a;\beta_t) - \sum_{b\in A}\mu(b|i_t;Q_t)\nabla_{\beta_t}Q_t(b;\beta_t)], \qquad (5.2)$$

where τ is the Boltzmann temperature (see Neu and Szepesvári, 2007). Thus, computing $\nabla_{\beta_t}\mu(a|i_t;Q_t)$ reduces to computing $\nabla_{\beta_t}Q_t(a;\beta_t)$.

The value of Q_t is generated by the finite-horizon planning algorithm. However, as I present below, the process of computing the gradient closely resembles the process of planning itself, and the two computations can be interleaved. Theorem 5.1 presented below is an adaptation of Proposition 4 from Neu and Szepesvári (2007). It presents the gradient computation for finite horizon planning as well as for infinite-horizon planning.

The theorem follows directly from Proposition 4 of Neu and Szepesvári (2007), and is therefore presented without proof. The theorem requires an assumption on the form of the reward function parameterization.

Assumption 5.1. The gradient of the reward function with respect to the parameters exists and is bounded: $\sup_{\beta,o,i,a}\|\nabla_\beta R(i,o,a;\beta)\| < \infty$.

Theorem 5.1. *Except on a set of measure zero, for any horizon H, the gradient*

Algorithm 2: PGRD (Policy Gradient for Reward Design) Algorithm

Input: T, β_0, $\{\alpha_t\}_{t=0}^{\infty}$, ξ, γ

1 $o_0, i_0 =$ `initializeStart()`
2 **for** $t = 0, 1, 2, 3, \ldots$ **do**
3 $\quad \forall_a Q_t(a;\beta_t), \nabla_{\beta_t} Q_t(a;\beta_t) \leftarrow$ `plan`$(i_t, o_t, T, R(i_t, \cdot, \cdot; \beta_t), d, \gamma)$
4 $\quad a_t \sim \mu(a|i_t; Q_t)$
5 $\quad r_{t+1}, o_{t+1} =$ `takeAction`(a_t)
6 $\quad z_{t+1} = \xi z_t + \tau[\nabla_{\beta_t} Q_t(a;\beta_t) - \sum_{b \in A} \mu(b|i_t; Q_t) \nabla_{\beta_t} Q_t(b;\beta_t)]$
7 $\quad \beta_{t+1} = \beta_t + \alpha_t(r_{t+1} z_{t+1} - \lambda \beta_t)$
8 $\quad i_{t+1} =$ `updateInternalState`(i_t, a_t, o_{t+1})
9 **end**

$\nabla_\beta Q^H(o, a; \beta)$ exists and is given by the recursion[4]

$$\nabla_\beta Q^H(o, a; \beta) = \nabla_\beta R(o, a; \beta) + \gamma \sum_{o' \in O} T(o'|o, a) \sum_{b \in \mathcal{A}} \pi^{H-1}(b|o') \nabla_\beta Q^{H-1}(o', b; \beta), \tag{5.3}$$

where $\nabla_\beta Q^1(o, a; \beta) = \nabla_\beta R(o, a; \beta)$ and $\pi^H(a|o) \in \arg\max_a Q^H(o, a; \beta)$ is any policy that is greedy with respect to Q^H. The result also holds for $\nabla_\beta Q^*(o, a; \beta) = \nabla_\beta \lim_{d \to \infty} Q^H(o, a; \beta)$.

The Q-function will not be differentiable when there are multiple optimal policies. This is reflected in the arbitrary choice of π in the gradient calculation. However, it was shown by Neu and Szepesvári (2007) that even for values of β which are not differentiable, the above computation produces a valid calculation of a subgradient; I discuss this below in the proof of convergence of PGRD.

5.1.3 Algorithm

PGRD (Algorithm 2) follows the form of OLPOMDP (Algorithm 1 of Bartlett and Baxter 2000) but generalizes it in two places. In Algorithm 2 line 3, the agent plans to compute the soft-max parameters Q_t, rather than storing the policy directly. It also simultaneously computes the gradient $\nabla_\beta Q^H(i, o, a; \beta)$, as given by equation (5.3). In line 8, the agent maintains a general notion of internal state that allows for a richer parameterization of policies than typically considered (similar to Aberdeen and Baxter, 2002). In the remaining lines, an action from the stochastic planned policy

[4] I have dropped the dependence on i in the theorem statement for simplicity.

μ is sampled and taken in lines 3 and 4, and the reward parameters are updated according to the OLPOMDP algorithm in lines 6 and 7.

The algorithm takes as parameters a sequence of learning rates $\{\alpha_k\}$, a parameter $\xi \in [0, 1)$ which biases the gradient calculation towards recent observations, and regularization parameter $\lambda > 0$ which keeps the reward parameters β bounded throughout learning. Given a sequence of calculations of the gradient of the policy with respect to the parameters, $\nabla_{\beta_t}\mu(a_t|i_t; Q_t)$, the remainder of the algorithm climbs the gradient of objective return $\nabla_\beta \mathbb{E}\left[U_{R_O}(h)|h \sim M\langle G(R(\cdot;\beta))\rangle\right]$ using OLPOMDP machinery. In the next subsection, I discuss how to compute $\nabla_{\beta_t}\mu(a_t|i_t; Q_t)$.

Computational Complexity

The most expensive line in Algorithm 2 is line 3, which computes the Q values and the gradient of the Q-values. The most direct method to calculate the Q-values in equation (5.1) is to directly unroll the recursive equations, effectively building a tree structure. This tree has size $(|\mathcal{S}||\mathcal{A}|)^H$, where $|\mathcal{S}|$ and $|\mathcal{A}|$ are the size of the state and action sets and H is the planning horizon. Thus, this simple planning algorithm is polynomial in state and action set sizes and exponential in horizon. An alternative is to implement the algorithm using H sweeps of value iteration (Sutton and Barto, 1998), which can be more efficient if the number of states is small. Using this alternative method, the complexity is polynomial in H. In the next chapter, I introduce more efficient methods that have a per-step complexity that is independent of the size of the state-space.

Here, I address the computational costs of the gradient computation in PGRD, the most expensive step in the PGRD algorithm. Computing the gradient of the Q-values with respect to β is similar to computing the Q-values, because equation (5.3) is similar to equation (5.1). The main difference is that whereas the outputs of $R(i, o, a; \beta)$ and $Q(i, o, a; \beta)$ are scalars, the outputs of $\nabla_\beta R(i, o, a; \beta)$ and $\nabla_\beta Q(i, o, a; \beta)$ are vectors. Thus the dependence on $|\mathcal{S}|$, $|\mathcal{A}|$, and H is the same for both computations. It is possible to compute the gradient both using a tree-style algorithm or a value-iteration style algorithm.

The gradient computational cost depends on the number of reward function parameters (the length of the gradient vector) in two places: in the computation of the reward function $R(i, o, a; \beta)$, and in the computation of the gradient of the reward function with respect to the reward parameters $\nabla_{\beta_t} R(i, o, a; \beta)$. Assume that the computation of the reward function and its gradient is linear in time and space in the number of reward parameters. This is the case when using a linear parameterization—a

dot product—as I do in all of the empirical work in this dissertation. The computation of $\nabla_\beta Q(i, o, a; \beta)$ is affected in two ways. First, increasing the number of parameters increases the complexity of computing $\nabla_\beta R(i, o, a; \beta)$ (linearly by assumption). Second, in the tree-based algorithm for example, it also increases the cost of the building the tree itself, because the tree nodes must store vectors instead of scalars. Fortunately, this cost increase is linear (time and memory) in the number of reward parameters as well.

The cost of the internal state update in line 8 is agent-dependent and is therefore ignored in this analysis. In some agents, such as ones that keep Markov representations, this update is cheap. In others that store statistics of history, such as POMDP belief-state representations, this step can be expensive.

5.1.4 Convergence of PGRD

Given a particular fixed reward function $R(\cdot; \beta)$, transition model T, and planning horizon, there is a corresponding fixed randomized policy $\mu(a|i; \beta)$. Denote the agent's internal-state update as a (usually deterministic) distribution $\psi(i'|i, a, o)$. Given a fixed reward parameter vector β, one can model transitions in the joint environment and internal state space as a Markov chain with a $|\mathcal{S}||I| \times |\mathcal{S}||I|$ transition matrix $\mathbf{P}(\beta)$ whose entries are given by $\mathbf{P}_{\langle s,i\rangle,\langle s',i'\rangle}(\beta) = p(\langle s', i'\rangle|\langle s, i\rangle; \beta) = \sum_{o,a} \psi(i'|i, a, o)\Omega(o|s')P(s'|s, a)\mu(a|i; \beta)$. I make the following assumptions about the agent and the environment:

Assumption 5.2. The transition matrix $\mathbf{P}(\beta)$ of the Markov chain with the joint environment and internal state space has a unique stationary distribution $\pi(\beta) = [\pi_{s_1,i_1}(\beta), \pi_{s_1,i_2}(\beta), \ldots, \pi_{s_{|\mathcal{S}|},i_{|I|}}(\beta)]$ satisfying the balance equations $\pi(\beta)\mathbf{P}(\beta) = \pi(\beta)$, for all $\beta \in \mathcal{B}$.

Assumption 5.3. During its execution, with probability 1, PGRD (Algorithm 2) does not reach a value of i_t, and β_t at which $Q^H(i_t, o, a; \beta_t)$ is not differentiable with respect to β_t, $\forall o \in O, a \in A$.

It follows from Assumption 5.2 that the expected objective return of the agent, $\mathbb{E}\left[U_{R_O}(h)|h \sim M\langle G(R(\cdot; \beta))\rangle\right]$, is independent of the start state. The original convergence proof for OLPOMDP (Bartlett and Baxter, 2000) has a similar condition that only considers environment states. Intuitively, Assumption 5.2 allows PGRD to handle history-dependence of a reward function in the same manner that it handles partial observability in an environment. Assumption 5.3 accounts for the fact that a

planning algorithm may not be fully differentiable everywhere. However, Theorem 5.1 showed that infinite and bounded-horizon planning is differentiable almost everywhere (in a measure theoretic sense). Furthermore, this assumption is perhaps stronger than necessary, as stochastic approximation algorithms, which provide the theory upon which OLPOMDP is based, have been shown to converge using subgradients (Kushner and Yin, 2010).

In order to state the convergence theorem, I must define the approximate gradient which OLPOMDP calculates. Let the approximate gradient estimate be $\overline{\nabla}_\beta \mathbb{E}\left[U_{R_O}(h)|h \sim M\langle G(R(\cdot;\beta))\rangle\right] \stackrel{\text{def}}{=} \lim_{T\to\infty} \sum_{t=1}^{T} r_t z_t$ for a fixed β and PGRD parameter ξ, where z_t (in Algorithm 2) represents a time-decaying average of the $\nabla_{\beta_t}\mu(a_t|i_t, Q_t)$ calculations. It was shown by Bartlett and Baxter (2000) that the approximate gradient $\overline{\nabla}_\beta$ is close to the true gradient ∇_β for large values of the ξ in OLPOMDP. Theorem 5.2 proves that PGRD converges to a stable equilibrium point based on this approximate gradient. This equilibrium point will typically correspond to some local optimum of the expected return $\mathbb{E}\left[U_{R_O}(h)|h \sim M\langle G(R(\cdot;\beta))\rangle\right]$. The theorem is a straightforward extension of Theorem 6 from Bartlett and Baxter (2000).

Theorem 5.2. *Given $\xi \in [0, 1)$, $\lambda > 0$, and a sequence of step sizes α_t satisfying $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} (\alpha_t)^2 < \infty$, PGRD produces a sequence of reward parameters β_t such that $\beta_t \to L$ as $t \to \infty$ a.s., where L is the set of stable equilibrium points of the differential equation $\frac{\partial \beta}{\partial t} = \overline{\nabla}_\beta \mathbb{E}\left[U_{R_O}(h)|h \sim M\langle G(R(\cdot;\beta))\rangle\right] - \lambda\beta$.*

Proof. (sketch) Bartlett and Baxter (2000) make 3 assumptions in their theorem. Their Assumption 1 is extended by Assumption 5.2 here. Assumption 5.2 allows the agent's internal state to be treated as part of the environment state in Bartlett and Baxter's proof (as for Aberdeen and Baxter, 2002). Assumption 2 from Bartlett and Baxter is satisfied by the assumption that the reward is defined over finite sets (an assumption they also make). It follows from assumption 5.1 that $\nabla_\beta Q_t(a|i_t;\beta_t)$ is bounded. Therefore, PGRD's choice of softmax to convert Q into a policy implies Assumption 3 from Bartlett and Baxter, hence I do not make that assumption here. The additional assumption added by PGRD, Assumption 5.3, is sufficient to guarantee that the non-differentiability of Q at some values of β does not affect the evolution of the algorithm. The remainder of the proof is a direct application of Theorem 6 from Bartlett and Baxter (2000). □

5.1.5 PGRD generalizes OLPOMDP

As stated above, OLPOMDP is a special case of PGRD when the planning horizon is one. This holds when OLPOMDP uses a Boltzmann (softmax) distribution parameterized with an action scoring function as its policy representation (a common case).

First, notice that in the case of horizon-1 planning, $Q^1(i, o, a; \beta) = R(i, o, a; \beta)$, regardless of the transition model and reward parameterization. Therefore, $\nabla_\beta Q^1(i, o, a; \beta) = \nabla_\beta R(i, o, a; \beta)$. Because $R(i, o, a; \beta)$ can be parameterized arbitrarily, PGRD can be configured to be identical to standard OLPOMDP with any policy parameterization that also computes an action scoring function for the Boltzmann distribution.

In the experiments, I demonstrate that choosing a planning horizon $H > 1$ can be beneficial over using OLPOMDP ($H = 1$). In the remainder of this section, I show theoretically that choosing $H > 1$ does not hurt in the sense that it does not reduce the space of policies available to the policy gradient method. Specifically, I show that when using an expressive enough reward parameterization, PGRD's space of policies is not restricted relative to OLPOMDP's space of policies. I prove the result for infinite-horizon planning, but the extension to finite-horizon planning is straightforward.

Theorem 5.3. *There exists a reward parameterization such that, for an arbitrary transition model T, the space of policies representable by PGRD with infinite-horizon planning is identical to the space of policies representable by PGRD with horizon 1 planning (standard OLPOMDP).*

Proof. I prove the result for a fixed internal state (holding it constant), though it is easily extended to the more general case. Let $C(o, a)$ be an arbitrary reward function used by PGRD with horizon 1 planning. Let $R(o, a; \beta)$ be a reward function for PGRD with infinite ($H = \infty$) planning. The horizon-∞ agent uses the planning result $Q^*(o, a; \beta)$ to act, while the horizon-1 agent uses the function $C(o, a)$ to act. Therefore, it suffices to show that one can always choose β such that the planning solution $Q^*(o, a; \beta)$ equals $C(o, a)$. This can be achieved by setting

$$R(o, a; \beta) = C(o, a) - \gamma \sum_{o'} T(o'|o, a) \max_{a'} C(o', a'),$$

for all $o \in O, a \in \mathcal{A}$. Substituting Q^* for C, this is the Bellman optimality equation (Sutton and Barto, 1998) for infinite-horizon planning. Setting $R(o, a; \beta)$ as above

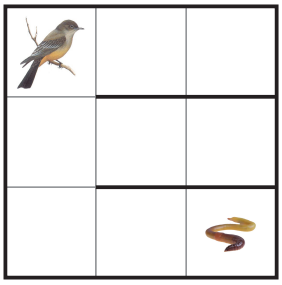


Figure 5.1: Foraging World

is possible if it is parameterized by a table with an entry for each observation–action pair. □

Theorem 5.3 also shows that the effect of an arbitrarily incorrect model can be overcome with an optimal reward function. This is because a Boltzmann distribution can, allowing for an arbitrary scoring function C, represent any policy. Thus, PGRD can be used to overcome a poor model (though gradient ascent methods are not guaranteed to find the global optimum). I demonstrate this ability of PGRD in the experiments.

5.2 Experiments

The primary objective of the experiments is to demonstrate that PGRD is able to use experience online to improve the reward function parameters, thereby improving the agent's expected objective return. Specifically, I compare the objective return achieved by PGRD to the objective return achieved by PGRD with the reward adaptation turned off (setting $\alpha_t = 0$ for all t). Unless otherwise noted, the reward function is initialized to the objective reward function. Thus, running PGRD with $\alpha = 0$ corresponds to an agent which uses the objective reward function. A secondary objective is to demonstrate that adding the ability to plan—even for small horizons—improves performance relative to the baseline algorithm of OLPOMDP (or equivalently PGRD with horizon $H = 1$), depending on the quality of the agent's model.

Importantly, PGRD is capable of improving reward functions in agents with limitations including the finite-horizon limitation and the use of a poor given or learned model. I demonstrate this by examining PGRD in similar agent-environment pairs to ones that were examined in Chapter 3.

Foraging Environment for Experiments 1 to 3. In this chapter, I use a slight modification of the foraging environment used in Chapter 3. In this case, it is stochastic, and the the bird does not have different hunger levels; the designer receives objective reward as soon as the agent eats the worm. This means that the agent is able to account for the effects of the objective reward with a planning horizon of 7, which is 1 fewer planning steps in this version of the environment than in the version in Chapter 3. This disparity due to historical reasons. In the next chapter, I demonstrate that PGRD is capable of improving planning agents with deeper planning horizons.

Specifically, the foraging environment illustrated in Figure 5.1 is a 3×3 grid world with 3 dead-end corridors (rows) separated by impassable walls. The agent (bird) has four available actions corresponding to each cardinal direction. Movement in the intended direction fails with probability 0.1, resulting in movement in a random direction. If the resulting direction is blocked by a wall or the boundary, the action results in no movement. There is a food source (worm) located in one of the three right-most locations at the end of each corridor. The agent has an `eat` action, which consumes the worm when the agent is at the worm's location. After the agent consumes the worm, a new worm appears randomly in one of the other two potential worm locations.

Objective Reward for the Foraging Environment. The designer's goal is to maximize the average number of worms eaten per time step. Thus, the objective reward function R_O provides a reward of 1.0 when the agent eats a worm, and a reward of 0 otherwise. The objective return is defined as the mean objective reward per step over an infinite horizon.

Experimental Methodology. I tested PGRD for Full Finite-Horizon Planning planning agents of horizons 1–7. Recall that PGRD for the agent with planning horizon 1 is the OLPOMDP algorithm. For each horizon , I jointly optimize over the PGRD algorithm parameters, α and ξ (I use a fixed α throughout learning). I test values for α on an approximate logarithmic scale in the range $(10^{-6}, 10^{-2})$ as well as the special value of $\alpha = 0$, which corresponds to an agent that does not adapt its reward function. I optimize over ξ values in the set $0, 0.4, 0.7, 0.9, 0.95, 0.99$. Following common practice (Baxter et al., 2001), I set the λ parameter to 0. I explicitly bound the reward parameters and capped the reward function output both to the range $[-1, 1]$. I used a Boltzmann temperature parameter of $\tau = 100$ and planning discount factor $\gamma = 0.95$. Because I initialized β_0 so that the initial reward function was the

objective reward function, PGRD with $\alpha = 0$ was equivalent to a conventional FFHP planning agent.

5.2.1 Experiment 1: A fully observable environment with a correct model learned online

In this experiment, I use PGRD to design the reward function in an agent whose only limitation is planning horizon, using (1) a general reward parameterization based on the current observation and (2) a more compact reward parameterization which also depends on the history of observations.

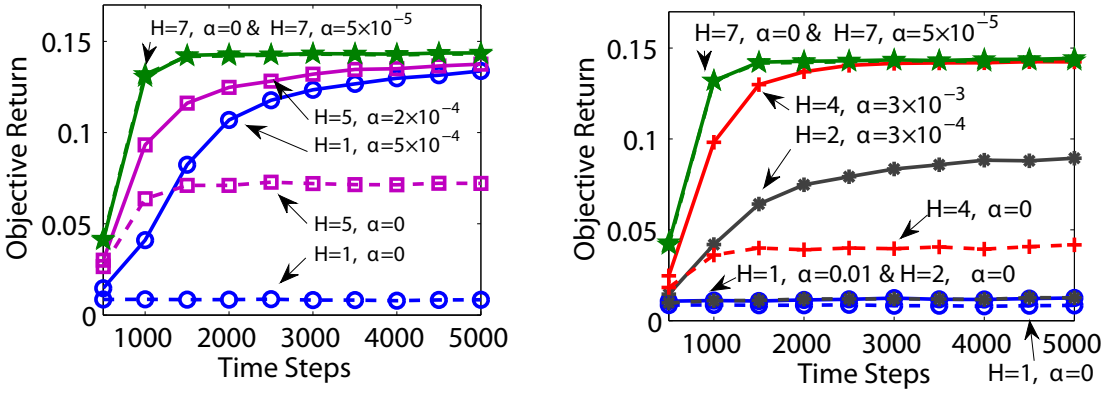
Observation. The agent observes the full state, which is given by the pair $o = (l, w)$, where o is the observation, l is the agent's location and w is the worm's location.

Learning a Correct Model. Although the theorem of convergence of PGRD relies on the agent having a fixed model, the algorithm itself is readily applied to the case of learning a model online. In this experiment, the agent's model T is learned online based on empirical transition probabilities between observations (recall this is a fully observable environment). Let $n_{o,a,o'}$ be the number of times that o' was reached after taking action a after observing o. The agent models the probability of seeing o' as $T(o'|o, a) = \frac{n_{o,a,o'}}{\sum_{o'} n_{o,a,o'}}$.

Reward Parameterizations. Recall that $R(i, o, a; \beta) = \beta^\intercal \phi(i, o, a)$, for some $\phi(i, o, a)$. I examine two reward parameterizations, which corresponds to two choices of ϕ.

(1) In the *observation-action parameterization*, $\phi(i, o, a)$ is a binary feature vector with one binary feature for each observation-action pair—internal state is ignored. This is effectively a table representation over all reward functions indexed by (o, a). As shown in Theorem 5.3, the observation-action feature representation is capable of producing arbitrary policies over the observations. In Experiment 1, ϕ is a 324-dimensional feature vector. Brute-force search over a 324-dimensional continuous space is infeasible.

(2) The *inverse-recency parameterization* is a more compact representation which uses features that rely on the history of observations. Its inverse-recency feature is similar to the inverse-recency feature in Chapter 3. Specifically,the inverse-recency



(a) Observation-action reward features

(b) Inverse-recency reward features

Figure 5.2: Performance of PGRD in the foraging environment with a correct learned model

feature in Chapter 3 was a function of state-action pair, whereas here the inverse-recency feature is a function of agent-location and/or action. The use of locations instead of states improves the performance of the reward feature in the foraging environment, because it more directly encodes a good exploration strategy for the agent. The feature vector is $\phi(i,o,a) = [R_O(o,a), 1, \phi_{c_l}(l,i), \phi_{c_{l,a}}(l,a,i)]$, where $R_O(o,a)$ is the objective reward function defined as above. The feature $\phi_{c_l}(l) = 1 - 1/c(l,i)$, where $c(l,i)$ is the number of time steps since the agent has visited location l, as represented in the agent's internal state i. Its value is normalized to the range $[0,1)$ and is high when the agent has not been to location l recently. The counts are initialized to a large number for each unvisited location. The feature $\phi_{c_{l,a}}(l,a,i) = 1 - 1/c(l,a,i)$ is similarly defined with respect to the time since the agent has taken action a in location l. The feature $\phi_{c_l}(l,i)$ is the feature I expect to get used by the agent, because it encodes how long it has been since the agent has looked for the food in that location. The feature $\phi_{c_{l,a}}(l,a,i)$ includes redundant information about whether the agent has tried each action in each location. I expect it to be ignored by PGRD, testing PGRD's ability to ignore poor reward features.

The use of these two feature sets allows for the comparison of an expressive feature set (observation-action) with a compact one (inverse-recency). This also allows for the comparison of a reward function that changes with history (inverse-recency) with one that doesn't (observation-action).

Results & Discussion. Figure 5.2(a) and Figure 5.2(b) present results for agents that use the observation-action parameterization and the inverse-recency parameterization of the reward function respectively. The horizontal axis is the number of time steps of experience. The vertical axis is the objective return, i.e., the average objective reward per time step. Each curve is an average over 130 trials. The values of H and the associated optimal algorithm parameters for each curve are noted in the figures. First, note that with $H = 7$, the agent is unbounded, because this is how many planning steps are required to account for the objective reward. Therefore, the agent $H = 7$ does not benefit from adapting the reward function parameters (given that I initialize to the objective reward function). Indeed, the $H = 7, \alpha = 0$ agent performs as well as the best reward-optimizing agent. The performance for $H = 7$ improves with experience because the model improves with experience (and thus from the curves it is seen that the model gets quite accurate in about 1500 time steps). The largest objective return obtained for $H = 7$ is also the best objective return that can be obtained for any value of H.

There are some commonalities that can be observed in both Figures 5.2(a) and 5.2(b). First, each curve that uses $\alpha > 0$ (solid lines) improves with experience. This is a demonstration that PGRD is able to effectively improve the reward function with experience. That the improvement over time is not just due to model learning is seen in the fact that for each value of $H < 7$ the curve for $\alpha > 0$ (solid-line) which adapts the reward parameters does significantly better than the corresponding curve for $\alpha = 0$ (dashed-line); the $\alpha = 0$ agents still learn the model. Second, for both $\alpha = 0$ and $\alpha > 0$ agents, the objective return obtained by agents with equivalent amounts of experience increases monotonically as H is increased (though to maintain readability I show selected values of H in each figure). This demonstrates that the ability to plan in PGRD significantly improves performance over standard OLPOMDP (PGRD with $H = 1$).

There are also some interesting differences between the results for the two different reward function parameterizations. With the observation-action parameterization, I noted that there always exists a setting of β for all H that will yield optimal objective return. This is seen in Figure 5.2(a) in that all solid-line curves approach optimal objective return. In contrast, the more compact inverse-recency reward parameterization does not afford this guarantee and indeed for small values of H (< 4), the solid-line curves in Figure 5.2(b) converge to less than optimal objective return. Notably, OLPOMDP ($H = 1$) does not perform well with this feature set. On the other hand, for planning horizons $4 \leq H < 7$, the PGRD agents with the

inverse-recency parameterization achieve near-optimal objective return faster than the corresponding PGRD agent with the observation-action parameterization. Finally, this experiment validates the claim that PGRD can improve reward functions that depend on the agent's history.

5.2.2 Experiment 2: A fully observable environment and poor given model

The theoretical analysis showed that PGRD with an incorrect model and the observation–action reward parameterization should (modulo local maxima issues) do just as well asymptotically as it would with a correct model. Here I illustrate this theoretical result empirically on the same foraging environment and objective reward function used in Experiment 1. I also test the hypothesis that a poor model should slow down the rate of learning relative to a correct model.

Poor Model. I gave the agents a fixed incorrect model of the foraging environment that assumes there are no internal walls separating the 3 corridors.

Reward Parameterization. I used the observation–action reward parameterization. With a poor model it is no longer interesting to initialize β so that the initial reward function is the objective reward function because even for $H = 7$ such an agent would do poorly. Furthermore, I found that this initialization leads to excessively bad exploration and therefore poor learning of how to modify the reward. Thus, I initialize β to uniform random values near 0, in the range $(-10^{-3}, 10^{-3})$.

Results. Figure 5.3 plots the objective return as a function of number of steps of experience. Each curve is an average over 36 trials. As hypothesized, the bad model slows learning by a factor of more than 10 (notice the difference in the x-axis scales from those in Figure 5.2). Here, deeper planning results in slower learning and indeed the $H = 1$ agent that does not use the model at all (OLPOMDP) learns the fastest. However, also as hypothesized, because they used the expressive observation–action parameterization, agents of all planning horizons mitigate the damage caused by the poor model and eventually converge to the optimal objective return.

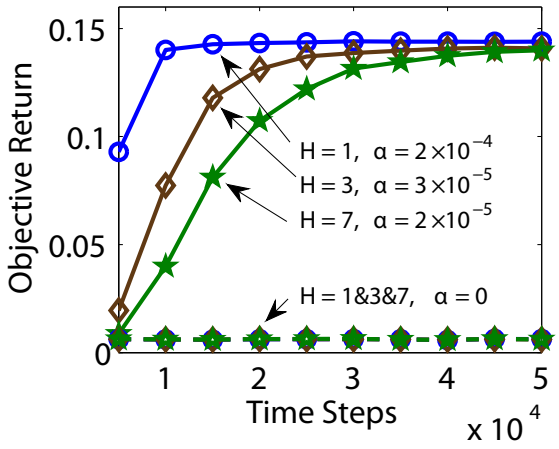


Figure 5.3: Performance of PGRD with a poor model

5.2.3 Experiment 3: Partially observable foraging world

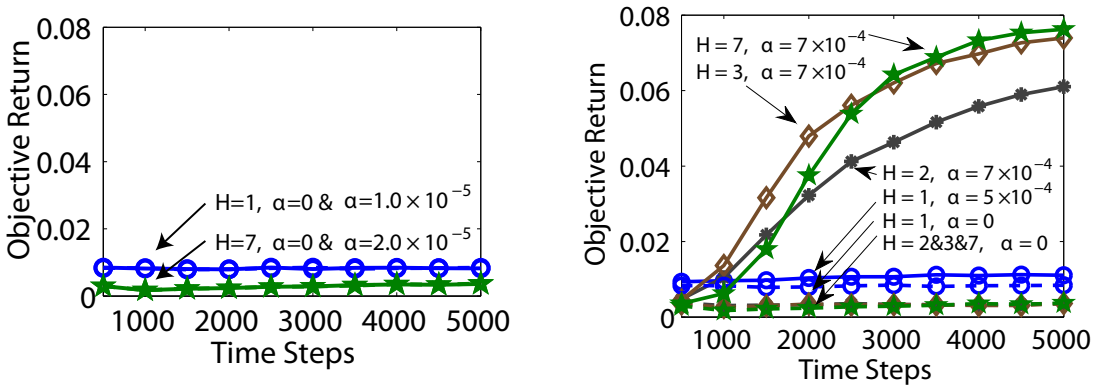
Here I evaluate PGRD's ability to learn in a partially observable version of the foraging domain. In addition, the agents learn a model under the erroneous (and computationally convenient) assumption that the domain is fully observable. This experiment mirrors Experiment 2 in Chapter 3. Though here, I examine the state representation limit and the finite-horizon limit simultaneously, while demonstrating that PGRD can optimize the reward functions online in this setting.

Partial Observation. Instead of viewing the location of the worm at all times, the agent can now see the worm only when it is colocated with it: its observation is $o = (l, f)$, where f indicates whether the agent is colocated with the food.

Learning an Incorrect Model. The model is learned just as in Experiment 1. Because of the erroneous full observability assumption, the model will hallucinate about worms at all the corridor ends based on the empirical frequency of having encountered them there.

Reward Parameterization. I use the observation-action and the inverse-recency reward parameterization. As in the first experiment, the parameters β are initialized such that the initial reward function equals the objective reward function.

Results. Each curve is the mean of 260 trials. Figure 5.4(a) plots the learning curves with the observation-action reward feature set. Due to the partial observability,



(a) Observation-action features

(b) Inverse-recency features

Figure 5.4: Performance of PGRD in partially observable foraging world

agents with the observation–action feature set, which does not contain information from history, perform poorly in this environment.

Figure 5.4(b) plots results from the experiment with the inverse-recency reward feature set. As seen in the solid-line curves, PGRD improves the objective return at all horizons (only a small amount for $H = 1$ and significantly more for $H > 1$). In fact, agents which don't adapt the reward are hurt by planning (relative to $H = 1$). This experiment demonstrates that the combination of planning and reward improvement can be beneficial even when the model is erroneous. Because of the partial observability, optimal behavior in this environment achieves less objective return than in Experiment 1.

5.2.4 Experiment 4: Acrobot

In this experiment I test PGRD in the Acrobot environment (Sutton and Barto, 1998), shown in Figure 5.5. It is a common benchmark task in RL, because it requires a non-trivial control policy and has a continuous state space, and it is one that has previously been used in the testing of policy gradient approaches (Weaver and Tao, 2001). This experiment demonstrates that PGRD can improve reward functions in an environment with a continuous state space. It further demonstrates that PGRD outperforms OLPOMDP.

Environment and Objective Reward. The dynamics of Acrobot are as specified by Sutton and Barto (Sutton and Barto, 1998). It is a two-link robot arm in which

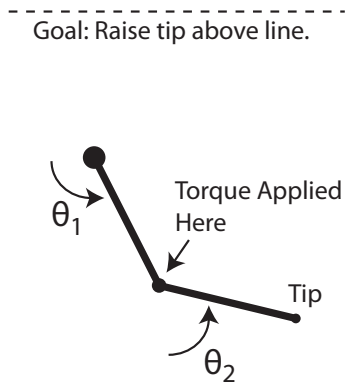


Figure 5.5: Acrobot Environment

the position of one shoulder-joint is fixed and the agent's control is limited to 3 actions which apply torque to the elbow-joint. The fully-observable state space is 4 dimensional, with two joint angles ψ_1 and ψ_2, and two joint velocities $\dot{\psi}_1$ and $\dot{\psi}_2$.

The designer receives an objective reward of 1.0 when the tip is one arm's length above the fixed shoulder-joint, after which the bot is reset to its initial resting position—hanging with the tip at its lowest point, with joint velocities of 0.

Model. I provide the agent with a perfect model of the environment. Because the environment is continuous, value iteration is intractable, and computational limitations prevent planning deep enough to compute the optimal action in any state. However, because the environment is deterministic, the FFHP algorithm is tractable because the next-state distribution is finite and small. The feature vector contains 13 entries. One feature corresponds to the objective reward signal. For each action, there are 5 features corresponding to each of the state features plus an additional feature representing the height of the tip: $\phi(i, o, a) = [R_O(o), \{\psi_1(o), \psi_2(o), \dot{\psi}_1(o), \dot{\psi}_2(o), h(o)\}_a]$. The height feature has been used in previous work as an alternative definition of objective reward (Weaver and Tao, 2001).

Results & Discussion. I plot the mean of 80 trials in Figure 5.6. Agents that use the fixed ($\alpha = 0$) objective reward function with bounded-horizon planning perform according to the bottom two curves. Allowing PGRD and OLPOMDP to adapt the parameters β leads to improved objective return, as seen in the top two curves in Figure 5.6. Finally, the PGRD $H = 7$ agent outperforms the standard OLPOMDP agent (PGRD with $H = 1$), further demonstrating that PGRD outperforms OLPOMDP.

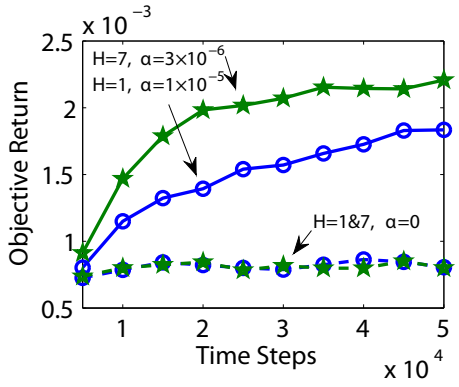


Figure 5.6: Performance of PGRD in Acrobot

5.3 Conclusion

In this chapter, I developed PGRD, a new method for approximately solving the optimal reward problem which can be applied in an online setting. I showed that PGRD is a generalization of OLPOMDP and demonstrated that it both improves reward functions in limited agents and outperforms the model-free OLPOMDP approach. In the next chapter, among other contributions, I extend PGRD to apply to Sparse Sampling and UCT, two algorithms which are designed for more efficiently planning in large state spaces.

Chapter 6

Optimal Reward Functions versus Leaf-Evaluation Heuristics[1]

The Optimal Reward Problem (ORP) approach is not the only method for overcoming planning agent limitations that works by modifying the agent's return function. In this chapter, I compare the ORP approach to two alternative approaches for modifying a planning agent's return estimates. One, the Potential-Based Reward Shaping approach, is a popular reward design approach. The other, the Leaf-Evaluation Heuristic approach, is a standard approach for overcoming the limitations arising from bounds on the horizon of a planning tree.

Potential-Based Reward Shaping (PBRS) is popular reward function design approach which was discussed in relation to the optimal reward problem in Section 2.3.1. This chapter compares the two formally. Recall that when using a potential-based reward function, an optimal policy is guaranteed to be optimal under the objective reward function. However, a limited agent may exhibit different behavior when using a potential-based reward function than when using the objective reward function. In contrast to the PBRS approach, the ORP approach does not necessarily restrict its class of reward functions. As forecast in Section 2.3.1, this chapter demonstrates that, due to this generality, the ORP approach can outperform the PBRS approach. In other words, potential-based reward functions are not necessarily optimal.

In a planning agent which makes the finite-horizon approximation, the *Leaf-Evaluation Heuristic* (LEH) adds a heuristic value to the estimated return at the leaf states of the planning tree (Shannon, 1950). If the expected return obtainable after the leaf state were known, then it could be used as the leaf-state return estimate to compensate for the missing subtree below the leaf. As this is not known in practice, some method must be employed to estimate the leaf-state values. Typically,

[1]This chapter presents material from Sorg et al. (2011). The Othello material is original to this dissertation.

domain-dependent offline estimates of leaf-node values are used, or they are learned from experience. This dissertation has focused on overcoming the finite-horizon approximation instead by solving the optimal reward problem.

The central claim of this chapter is that there exist environments and agents in which the ORP approach outperforms both the LEH and PBRS approaches. In fact, I show that for every leaf-evaluation heuristic, there exists a potential-based reward function which produces equivalent behavior, and vice versa. Thus, the LEH and PBRS approaches are effectively equivalent. Furthermore, I show that the preference orderings over policies achievable through reward function design subsume those of the set of leaf-evaluation heuristics (and therefore also potential-based reward functions). The key aspect of this generality is *path dependence*. Reward function design is able to modify the agent's return function in a way that depends on entire state-action trajectories, while a leaf-evaluation heuristic affects the return only as a function of the final state in the trajectory. As a result, in agents which make approximations in shallow portions of the tree, optimal reward functions can compensate for errors that the LEH and PBRS approaches cannot compensate for.

Of course, the nature of this benefit depends heavily on the planning algorithm chosen and I cannot consider all planning algorithms in this chapter. I focus on UCT (Kocsis and Szepesvári, 2006), a state-of-the art planning algorithm that has been used in many applications (Gelly and Silver, 2008; Finnsson and Björnsson, 2008). UCT makes several approximations that can be mitigated through solving the ORP: (1) it cannot generate unbounded-length trajectories, causing the *finite-horizon bias*; (2) it uses Monte-Carlo sampling to approximate return values, resulting in *sampling variance*; and (3) its early sample trajectories evaluate a suboptimal policy, a problem I refer to as *search control bias*.

To analyze the differing abilities of leaf-evaluation functions and optimal reward functions to compensate for these approximations, I additionally compare two other planning algorithms which share some of these approximations: Full Finite-Horizon Planning (FFHP), studied in Chapters 3 and 5—which only suffers from the finite-horizon—and Sparse Sampling (Kearns et al., 1999)—which makes the finite-horizon and sampling approximations. In a second contribution of this chapter, I extend PGRD to develop PGRD-SS and PGRD-UCT—PGRD applied to Sparse Sampling and UCT, respectively. These new algorithms enable reward functions to be optimized in agents whose computational complexity is independent of the size of the state space.

I demonstrate these claims in several experiments, culminating with the use of PGRD-UCT to design reward functions for a UCT agent in the game of Othello, a

two-player game with an estimated 10^{28} states (Allis, 1994).

6.1 Optimal Leaf-Evaluation Heuristics

To simplify notation, in this chapter I consider *Markov* environments M with a finite number of states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$. The dynamics are governed by a state-transition function $T(s'|s,a)$ that defines a distribution over next state s' conditioned on current state s and action a. Similar to the previous chapter, I assume that the agent's reward function $R(s,a,s';\beta)$, which maps state s, action a, and next-state s' tuples to reward, is parameterized by a vector of parameters $\beta \in \mathbb{R}^m$ for some number of parameters m and is differentiable in β.

A policy μ maps state s to distributions over actions $\mu(a|s)$. A value function V^μ is the expected return of following the policy μ given that the agent starts in a particular state: $V^\mu(s) = \mathbb{E}[U_R(h)|\mu, s_0 = s]$. An optimal policy μ^* maximizes expected return from the current state, i.e., $\mu^* = \arg\max_\mu V^\mu(s)$. The optimal value function is the value function of an optimal policy $V^* = V^{\mu^*}$. Local planning agents repeatedly estimate these quantities from a current state by building planning trees using a given or learned model $\hat{T}(s'|s,a)$, and selecting the action corresponding to a locally optimal policy.

Leaf-Evaluation Heuristic (LEH). Let $h[i,j] = s_i, a_i, s_{i+1}, a_{i+1}, ..., s_j$ denote the subsequence of trajectory h from time i to j. Given some finite horizon H, define the truncated return function $U_{R_O}(h[0,H]) = \sum_{i=0}^{H-1} \gamma^i R_O(s_i, a_i, s_{i+1})$. The truncated return ignores rewards after the horizon H.

The leaf-evaluation heuristic compensates for this finite-horizon bias in an agent's return estimates. A leaf-evaluation heuristic is a function $L(s)$ defined over the state space $s \in S$. When planning with truncated trajectories, an agent adds L to its return estimates as a function of the final state in the trajectory. Define the leaf-evaluation return to be

$$U^L_{R_O}(h[0,H]) = U_{R_O}(h[0,H]) + \gamma^H L(s_H). \tag{6.1}$$

$\mathbf{V}^*$ LEH. The value function of an optimal policy, V^*, is the traditional notion of the ideal leaf-evaluation heuristic (hereafter called the V^* leaf-evaluation heuristic). This corresponds to setting $L(s) = V^*(s)\,\forall s$. The $\mathbf{V}^*$ LEH ignores the agent architecture, in that it is based on the value function of the optimal policy, which is a property of the environment and the objective reward function. However, using this function

completely corrects for the finite-horizon bias of the return estimate of an optimal policy when the agent computes full expectations over the truncated return. I use the V^* LEH as a baseline in the empirical work.

Optimal LEH. If the agent cannot exactly compute the full expectation over the truncated trajectories, the V^*-leaf-evaluation heuristic may not be optimal. This can happen if the agent makes approximations in addition to the finite-horizon approximation, as is usual in planning methods applicable to real domains. An optimal LEH is defined similarly to an optimal reward function:

$$L^* = \arg\max_{L \in \mathcal{L}} \mathbb{E}\left[U_{R_O}(h) | h \sim M\langle G(U^L_{R_O})\rangle\right],$$

where the distribution over h is the result of agent G planning with leaf-evaluation return function $U^L_{R_O}$ acting in environment M, and $\mathcal{L}$ is the set of all mappings from S to $\mathbb{R}$. Note that I am passing the return function in as an agent parameter, as I formerly did with the reward function. For comparison, the optimal reward problem can be expressed this way as well:

$$R^* = \arg\max_{R \in R} \mathbb{E}\left[U_{R_O}(h) | h \sim M\langle G(U_R)\rangle\right],$$

where U_R is the agent's return function when using reward function R.

An optimal LEH leads to an agent with the best expected objective return for the designer. As for an optimal reward function, an optimal leaf-evaluation heuristic depends on the agent architecture. Below, I present a method for finding approximately optimal leaf-evaluation heuristics.

6.2 Reward Design Subsumes Leaf-Evaluation Heuristics

In this section, I prove that for any leaf-evaluation heuristic, there exists a reward function that produces identical behavior, but the converse is not true. The analysis is most closely related to the work of Asmuth et al. (2008) on potential-based reward shaping in model-based agents. Specifically, applying an evaluation function at the leaf is equivalent to adding a potential-based shaping function to the reward function, minus a constant offset depending on start state.

Define a potential-based reward function to be $R_L(s, a, s') = R_O(s, a, s') + \gamma L(s') -$

$L(s)$ for some function of state $L(s)$. Then, the following lemma holds:

Lemma 6.1. *For any history h and for all finite horizons H, the leaf-evaluation return $U^L_{R_O}(h[0,H])$ and the potential-based return $U_{R_L}(h[0,H])$ differ by a constant dependent only on start state.*

Proof. The additive reward bonus creates a telescoping sum:

$$\begin{aligned}
&U_{R_L}(h[0,H]) \\
&= \sum_{t=0}^{H-1} \gamma^t R_O(s_t, a_t, s_{t+1}) + \sum_{t=1}^{H} \gamma^t L(s_t) - \sum_{t=0}^{H-1} \gamma^t L(s_t) \\
&= U_{R_O}(h[0,H]) + \gamma^H L(s_H) - L(s_0) \\
&= U^L_{R_O}(h[0,H]) - L(s_0)
\end{aligned}$$

□

Lemma 6.1 states that for any leaf-evaluation heuristic L, there exists an effectively equivalent potential-based reward function R_L and vice versa. Specifically, the constant offset $L(s_0)$ will not affect an agent's preference ordering over histories. Furthermore, because policies are evaluated by taking expectations over histories, this constant offset will not affect an agent's preference ordering over policies. This does not guarantee that a particular approximate planning algorithm will return identical policies when planning with U_{R_L} versus planning with $U^L_{R_O}$. However, for the algorithms used in this chapter, it can easily be shown that planning with U_{R_L} and $U^L_{R_O}$ do indeed lead to identical behavior.

Lemma 6.1 shows that the set of reward functions effectively subsumes the set of leaf-evaluation functions, in the sense that they are capable of specifying at least as broad a set of preference orderings over policies. In Theorem 6.1 below, I prove that the relationship is strict; there exist reward functions that express preference orderings over policies which cannot be expressed via a leaf-evaluation heuristic.

Theorem 6.1. *The set of potential-based return functions is a strict subset of the set of reward-based return functions.*

Proof. Lemma 6.1 proves non-strict subsumption. Here, I prove strictness. Consider two trajectories: $h_1 = (s_1, a_1, s_2, a_2, s_3)$ and $h_2 = (s_1, a_1, s_4, a_2, s_3)$, where the subscripts in this proof only indicate identity, not time. Let $U_{R_O}(h_1) = U_{R_O}(h_2) = 0$. Then, $U_{R_L}(h_1) = U_{R_L}(h_2) = -L(s_1) + \gamma^2 L(s_3)$ regardless of L. However, in general, $U_R(h_1) = R(s_1, a_1, s_2) + \gamma R(s_2, a_2, s_3) \neq U_R(h_2) = R(s_1, a_1, s_4) + \gamma R(s_4, a_2, s_3)$. □

The proof highlights the difference between general reward functions and both leaf-evaluation heuristics and potential-based reward functions. General reward functions allow for *path dependent* modifications of objective return—they depend on the entire state-action sequence in the trajectory—while leaf-evaluation functions and potential-based reward functions only modify the return as a function of the leaf state.

This result also shows that it is not beneficial to jointly optimize over leaf-evaluation heuristics and reward functions. Because reward functions effectively subsume leaf evaluation heuristics, one would be able to achieve just as strong an agent by optimizing only over reward functions.

6.3 Optimal Rewards Outperform Leaf-Values

In this section, I discuss how the extra generality afforded by reward functions can be beneficial in UCT by first discussing two other planning algorithms: Full Finite-Horizon Planning and Sparse Sampling. This progression enables the isolation of different approximations made by UCT. For each approximation, I discuss whether the generality of reward design can lead to better performance over leaf-evaluation heuristics.

6.3.1 Full Finite-Horizon Planning

FFHP is the simple example of a computationally limited planning agent previously used in Chapters 3 and 5. Each time step, FFHP computes a finite-horizon approximation of the full planning problem. Using the notation in this chapter, Let $Q^{\mu,H}$ be the expected truncated return of policy μ given that the agent follows μ after taking action a from state s: $Q^{\mu,H}(s,a) = \mathbb{E}[U_R(h[0,H])|\mu, s_0 = s, a_0 = a]$. FFHP computes value function of the optimal policy under the truncated return $Q^H(s,a) = \max_\mu Q^{\mu,H}(s,a)\, \forall a \in \mathcal{A}$ and acts greedily with respect to $Q^H(s,a)$. There are numerous ways to calculate $Q^H(s,a)$; one way is to build a finite-horizon, balanced planning tree rooted at the current state. For each state-action node, the tree contains a child node for all possible successor states and an entry describing the probability of reaching each state under the model.

FFHP computes the true expectation of the truncated return; therefore, as discussed above, the V^*-leaf-evaluation heuristic perfectly corrects for the approximation. Thus, the optimal reward function R^* will do no better than the optimal leaf-evaluation

heuristic L^* (or potential-based reward function) in FFHP. In other words optimal reward functions do not outperform leaf-evaluation heuristics on an agent which suffers only from the finite-horizon bias.

6.3.2 Sparse Sampling

FFHP is intractable in practice because the number of next states from each node can be large or infinite and because the size of the tree is exponential in the horizon. To overcome the first concern, Kearns et al. (1999) proposed Sparse Sampling, which approximates a FFHP planning tree by sampling the next-state dynamics. It builds a balanced, approximate FFHP planning tree by sampling C next-states for each state-action node. I present pseudo-code for Sparse Sampling in Section 6.4.1 below, along with code for computing the gradient of its behavior with respect to the reward parameters.

The sample tree built by Sparse Sampling causes two types of errors not present in FFHP when estimating the action-value function at the root state: (1) the iterated sampling of C next states in building the tree introduces a variance into the action-value estimates; and (2) in the backing up of values from the leaves of the tree to the root, the max operation over noisy estimates introduces a positive bias into the action-value estimates. Crucially these errors occur throughout the tree, and indeed increasing the horizon compounds both sources of errors. Thus, leaf-evaluation heuristics, which are functions of the leaf state only, are less able to mitigate these errors relative to designed reward functions, which can have effects at all depths in the tree. To appreciate this difference, consider that the V^*-leaf-evaluation heuristic, in particular, is entirely about accounting for the expected value of the missing subtree *below* the leaf state. While effective at mitigating the finite-horizon bias common to FFHP and Sparse Sampling, it has little or no correlation to the factors inducing the additional errors in the tree *above* the leaf state. The empirical results below confirm that not only can optimal reward functions be more effective than the V^*-LEH, but they can also be more effective than optimal leaf-evaluations heuristics (L^*)—that is, even when allowing leaf evaluation heuristics to mitigate errors introduced by the sampling approximations to the greatest extent possible.

6.3.3 UCT

Even Sparse Sampling, however, is infeasible on practical problems, because the expense of building the tree is exponential in the horizon. UCT (Kocsis and Szepesvári, 2006) does not build a balanced tree. Instead, it samples N, H-length trajectories from the current state, constructing an imbalanced tree from the sampled trajectories. This results in the challenge of choosing sample actions, a problem known as search control.

UCT chooses actions using statistics from previous sample trajectories. It estimates the value of a state, action, depth tuple (s, a, d) as the average return obtained after experiencing the tuple:

$$Q(s,a,d) = \sum_{i=1}^{N} \frac{I_i(s,a,d)}{n(s,a,d)} \sum_{k=d}^{H-1} \gamma^{k-d} R(s_k^i, a_k^i, s_{k+1}^i; \beta),$$

where $n(s,a,d)$ is the number of times tuple (s,a,d) has been sampled, $I_i(s,a,d)$ is 1 if tuple (s,a,d) is in trajectory i and 0 otherwise, s_k^i is the k^{th} state in the i^{th} trajectory, and a_k^i is the k^{th} action in the i^{th} trajectory. UCT selects search control actions using Q and an additional bonus[2] which encourages balanced sampling. Prior to having sampled all actions for a given state–depth pair, an action is sampled uniformly randomly among those not yet explored. After all actions have been tried at least once, the agent chooses

$$a^* = \arg\max_a Q(s,a,d) + c\sqrt{\frac{\log n(s,d)}{n(s,a,d)}},$$

where $n(s,d) = \sum_a n(s,a,d)$. With an appropriate choice of c, UCT's Q estimates at the root converge to FFHP's Q estimates as the number of trajectories N increases.

For finite N, UCT shares the finite depth and sample variance errors from Sparse Sampling, but it also introduces another source of error in the estimation of action values at the root. During initial trajectory generation, the algorithm incorporates suboptimal actions into its evaluation of the optimal policy, producing an error in the value estimates which I refer to as the *search control bias*. This error in UCT occurs throughout the trajectories. Thus, I expect that optimal reward functions, which can modify the calculation of return based on all the state-action pairs along a trajectory, will be better at mitigating this error relative to leaf evaluation heuristics, which can

[2]I emphasize that the UCT search control bonus is not a reward, because it does not directly impact external action selection, and the agent does not consider the effect of multiple search-control bonuses in sequence.

modify the calculation of return based only on the leaf state of the trajectory. The experimental results below confirm this intuition.

6.4 Reward Design in Sample-based Planners

In the previous chapter, I showed how to compute the subgradient of FFHP's policy with respect to the reward parameters. In order to extend Sparse Sampling and UCT to have their reward functions improved by PGRD, I need to develop similar subgradient computations for these approaches. The first step is to modify the algorithms so that they select actions stochastically, given their computed Q-values. Much like PGRD, developed in the previous chapter, the PGRD-SS and PGRD-UCT methods developed in this chapter plan using the algorithms above, but execute actions according to a softmax distribution at the root of the tree. Specifically, $\mu(a|s) \stackrel{\text{def}}{=} \frac{e^{\tau Q(s,a)}}{\sum_b e^{\tau Q(s,b)}}$, where τ is a temperature parameter controlling how deterministically the agent selects the action with the highest score and $Q(s,a)$ is the action-value computed by the planning tree rooted at state s.

6.4.1 Reward Improvement in Sparse Sampling

Unlike in FFHP, there are two sources of stochasticity in Sparse Sampling. In addition to the stochasticity created by the softmax action selection, the tree sampling creates stochasticity in the computed Q values. I denote the distribution over Q-values $p(Q|s;\beta)$. Combining these two sources of randomness produces the full distribution over next action: $\mu(a|s;\beta) = \sum_Q \mu(a|s,Q)p(Q|s;\beta)$. I do not have an analytical formula for $p(Q|s;\beta)$. Estimating it is prohibitively expensive, requiring many sample trees.

This problem can be solved by noting that the distribution over planning trees is independent of the reward parameters. Thus, the Sparse Sampling algorithm can be viewed as being composed of two stages. First, the algorithm samples a planning tree, then the reward function is applied to that tree to compute the Q-values. Let i denote a sampled planning tree. The stochastic policy can now be broken up as: $\mu(a|s;\beta) = \sum_Q \mu(a|Q) \sum_i p(Q|s,i;\beta)p(i|s)$.

I simplify this equation in two steps. First, note that the Q-function is deterministic given a fixed tree and return function. Second, note that the tree distribution, though dependent on root state, is independent of the reward distribution. In other words,

Algorithm 3: SparseSamplingGradient(s, β, C, H, L)

Result: $Q(a)$: sample estimates of $Q^H(s, a; \beta) \quad \forall a \in \mathcal{A}$
$\overrightarrow{dQ}(a)$: gradient of sample estimates: $\nabla_\beta Q^H(s, a; \beta)$, $\forall a \in \mathcal{A}$

if $H = 0$ **then**
  return $\forall a \in \mathcal{A}$, $Q(a) = L(s)$, $\overrightarrow{dQ}(a) = \vec{0}$
end
for $a \in \mathcal{A}$ **do**
  $Q(a) \leftarrow 0$
  $\overrightarrow{dQ}(a) \leftarrow \vec{0}$
  for $i = 1 \ldots C$ **do**
    $s' \sim \hat{T}(\cdot|s, a)$
    Q', $\overrightarrow{dQ}' \leftarrow$ `SparseSamplingGradient` $(s', \beta, C, H-1, L)$
    $a^* = \arg\max_{a' \in \mathcal{A}} Q'(a')$
    $Q(a)$ += $\frac{1}{C}\left(R(s, a, s'; \beta) + \gamma Q'(a^*)\right)$
    $\overrightarrow{dQ}(a)$ += $\frac{1}{C}\left(\nabla_\beta R(s, a, s'; \beta) + \gamma \overrightarrow{dQ}'(a^*)\right)$
  end
end
return Q, dQ

the sparse sampling planning tree can be handled as if it were the internal state of the agent, as in the previous chapter (Chapter 5). Viewed this way, the agent acts according to $\mu(a|i; \beta)$, as in the previous chapter. The agent no longer needs to marginalize over planning trees; however, this simplification adds variance to the gradient estimate.

I present pseudo-code for the Sparse Sampling planning and gradient algorithm in Algorithm 3. The algorithm contains code for computing the estimates of Q and for simultaneously computing the gradient of these estimates with respect to the reward parameters. Each time a next-state s' is sampled in the inner-loop, the algorithm calculates the Q-estimates $Q(a)$ and gradient estimates $\overrightarrow{dQ}(a)$ from the sampled next state by recursively calling `SparseSamplingGradient` on the next state s' and backing up the Q estimates and the $\overrightarrow{dQ}$ estimates of the greedy action a^*.

PGRD-SS acts according to Sparse Sampling with the softmax action selection at the root, and computes the subgradient of the policy using equation 5.2 and Algorithm 3. PGRD-SS can be shown to be a convergent reward function optimization approach under the same assumptions as PGRD.

6.4.2 Reward Improvement in UCT

Like Sparse Sampling, PGRD-UCT has two sources of stochasticity when selecting actions. Unlike Sparse Sampling, however, the search control decisions depend on the reward function. Thus, instead of modeling the tree as part of the agent's internal state, PGRD-UCT uses an approximate solution which ignores the dependence on the reward function in the tree generation. This approximation is motivated by the proof of convergence of UCT. With an unlimited number of sample trajectories, the Q estimates converge to the FFHP estimates. Thus, in the limit, the search-control effects are negligible.

After this assumption is made, I can use the same trick used in Sparse Sampling—assume that the remaining stochasticity is conditioned solely on the environment. The resulting gradient computation is

$$\nabla_\beta Q(s,a,0) = \sum_{i=1}^{N} \frac{I_i(s,a,0)}{n(s,a,0)} \sum_{k=0}^{H-1} \gamma^k \nabla_\beta R(s_k^i, a_k^i, s_{k+1}^i; \beta). \tag{6.2}$$

Notice that the computation of the gradient resembles the computation of the Q estimates. Also, note that the gradient does not need to be computed at any depth d but $d = 0$. These two computations can be performed simultaneously, as the trajectories are generated. PGRD-UCT, plans with UCT, acts according to softmax action selection at the root, and computes the subgradient of the policy using equations 5.2 and 6.2.

6.5 Experiments

In this section, I empirically validate the claim that an approximately optimal reward function R^* can outperform an optimal leaf-evaluation heuristic L^* (and therefore all potential-based reward functions) in some agents and environments. The experiments also support the conclusion that an optimal leaf-evaluation heuristic L^* can be better than the V^*-leaf-evaluation heuristic.

I demonstrate these claims in two sets of environments. The first environment, Marble Maze, is a small environment that can be solved exactly, allowing for the V^* heuristic to be computed. Furthermore, its dynamics have properties that highlight the limitations inherent in Sparse Sampling and UCT. The second environment, Othello, is a two-player game with an estimated 10^{28} game states and an estimated game-tree

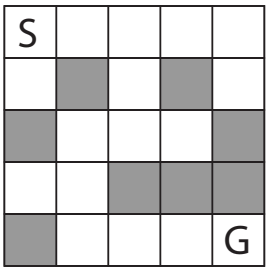


Figure 6.1: Marble Maze Environment

complexity of 10^{58} (Allis, 1994). In addition to providing more supporting evidence to the above claims, this final experiment demonstrates that PGRD-UCT can improve reward functions in problems that are too large for the optimal policy to be computed using current computational resources.

6.5.1 Marble Maze Experiments

The Marble Maze environment is pictured in Figure 6.1, a grid world based on a commonly-used set of noisy dynamics (Asmuth et al., 2008). The agent starts in the location marked S. The designer receives a reward of 1 when the agent reaches the goal state, marked G, after which the agent is transported back to the start. Otherwise, the designer receives 0 reward. From the pit states (gray squares), all actions lead the agent back to the start state. The agent has movement actions corresponding to the four cardinal directions. Each action has a 0.2 probability of failing; if the action fails, the agent instead moves in a random direction perpendicular to the intended direction, each with probability 0.5. In each trial, an agent is placed in a different instantiation of the Marble Maze environment, sampled uniformly randomly from all solvable 5x5 Marble Maze worlds with 8 pits and the start and goal locations in opposite corners.

The properties of Marble Maze highlight the deficiencies in Sparse Sampling and UCT. When a sample trajectory falls into a pit, it does great damage to the value of a trajectory. This creates a large return variance and a large search control bias when the agent's search control policy is poor.

Reward Function Design Spaces. Recall that using a potential-based reward function is equivalent to using a leaf-evaluation heuristic. Thus, by optimizing both the potential-based reward functions and more general reward functions, the PGRD algorithms can be used to compare approximately optimal leaf-evaluation heuristics

and approximately optimal reward functions. The general reward space I use in the Marble Maze experiments, $R(s, a, s'; \beta)$, is a look-up table over (s, a, s'). This is encoded using a linear parameterization $R(s, a, s'; \beta) = \phi(s, a, s)^{\mathsf{T}}\beta$, where ϕ is the unit-basis feature set. Specifically, $\phi(s, a, s')$ has one feature for every (s, a, s') pair, and each feature is 1 when the input matches a particular, unique (s, a, s') pair, and 0 otherwise.

The potential-based reward space is $R_L(s, a, s'; \beta) = R_O(s, a, s') - L(s; \beta) + \gamma L(s'; \beta)$, where $L(s; \beta)$ is a look-up table over s. The $L(s; \beta)$ look-up table is encoded analogously to the look-up table in the general reward case.

Experiment Methodology. For each planning algorithm, I plot PGRD learning curves for each reward space using various planning horizons. For each reward-space and for each planning-horizon, I optimize PGRD's learning rate parameter to a constant from a discrete set on a logarithmic scale in the range $[10^{-6}, 10^{-3}]$. Each learning curve uses the learning rate which yields the maximum expected cumulative objective reward. I used PGRD parameters $\beta = 0.9$ and $\lambda = 0$ and the softmax temperature $\tau = 100$. The reward parameters for both general reward functions and potential-based reward functions are initialized such that the functions are equal to the objective reward function R_O. The agents learn the transition models online using empirical counts: $\hat{T}(s'|s, a) = \frac{n(s,a,s')}{n(s,a)}$, where $n(\cdot)$ is the number of times the tuple has occurred in the agent's experience.

I use an additional baseline algorithm for learning potential-based reward functions, SARSA(λ) (Sutton and Barto, 1998), for comparison. The SARSA(λ) agent learns a table-based Q-function from the objective reward feedback and uses the maximum action-value in each state as the $L(s)$ value in the potential-based reward definition. The agent takes a random action with probability ϵ and the planning algorithm's action otherwise, planning with the learned potential-based reward function. I optimized over ϵ in the set $\{0, 0.01, 0.1, 0.2\}$. I optimized over learning rates from discrete values in the range $[0.01, 0.2]$ and used $\lambda = 0.95$. As opposed to the PGRD agent with a potential-based reward function, which attempts to find the equivalent of the L^*-LEH, the SARSA(λ) agent learns a value function and uses this as the LEH. In other words, the SARSA(λ) agent attempts to find the V^*-LEH. Finally, I also test the V^*-leaf-evaluation heuristic, found via value iteration.

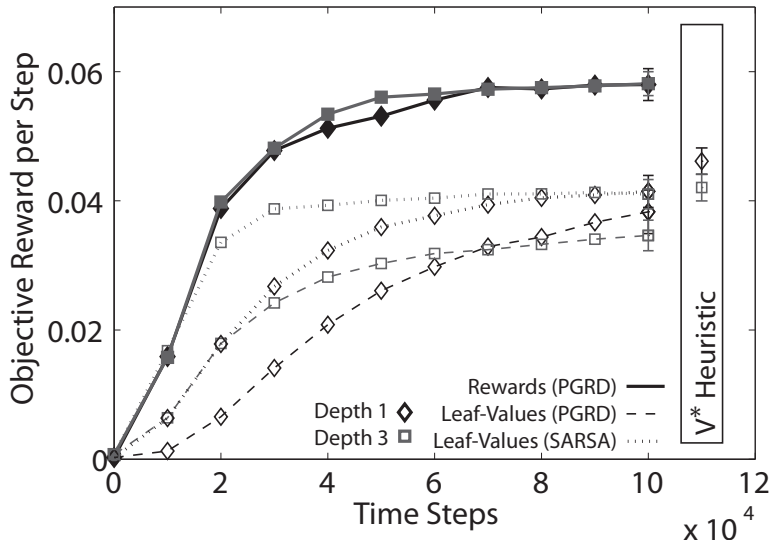


Figure 6.2: Performance of Sparse Sampling in Marble Maze. Solid-lines/filled markers indicate general reward functions. Dashed or Dotted-lines/hollow markers indicate leaf-evaluation functions. Inset on right: performance of V^*-leaf-evaluation functions, indicated by marker.

Sparse Sampling Experiment

In this section, I demonstrate that reward functions are better than leaf-evaluation heuristics at improving performance in agents with sampling approximation errors. Figure 6.2 presents learning curves for PGRD-SS agents with a fixed sample count of $C = 2$. I evaluated agents with planning horizons 1 through 4, though I show only horizons 1 and 3 for readability. The leftmost points on the curve represent agents which use the objective reward function. The remainder of the curve plots the objective reward per step obtained by PGRD-SS during optimization, averaged over the previous 10^4 time steps. At the final points in the curve I plot 95% confidence intervals. Each learning curve is an average of 100 trials.

The general reward functions (top 2 curves) outperform leaf-evaluation heuristics at each planning horizon, both for PGRD-optimized evaluation heuristics (bottom 2 curves) and SARSA-optimized evaluation heuristics (middle 2 curves). Although it might be difficult to see given the small difference in expected return, the leftmost points on the graph demonstrate that increasing the planning horizon improves the agent's performance for agents that use only the objective reward function R_O. This is because increasing the planning horizon reduces the finite-horizon approximation bias. Also as predicted, increasing the planning horizon hinders the leaf-value heuristic's

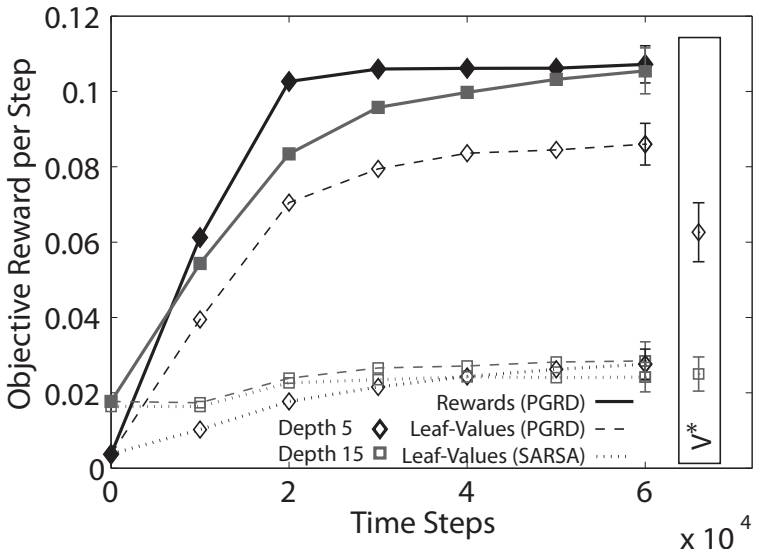


Figure 6.3: Performance of UCT in Deterministic Marble Maze. Solid-lines/filled markers indicate general reward functions. Dashed or Dotted-lines/hollow markers indicate leaf-evaluation functions. Inset on right: performance of V^*-leaf-evaluation functions, indicated by marker.

ability to compensate for the finite-horizon approximation. Thus, there is a cross-over effect. As a leaf-evaluation heuristic becomes more accurate, the optimal planning horizon decreases. This can be seen by the fact that the Horizon-1 V^*-LEH agent outperforms the Horizon-3 V^*-LEH agent. In contrast, the ability of optimal reward functions to induce path-dependent modifications of the objective return allows the reward-design agents to converge to near-optimal behavior (optimal behavior not shown) at each planning horizon.

UCT Experiment

In this section, I demonstrate that optimal reward functions are better than leaf-evaluation heuristics at mitigating the search-control bias. UCT has both the search control bias and sampling approximation errors (in addition to the finite-horizon bias). To separate the confounding sampling error explanation from the search-control bias explanation for the reward's improvement over leaf-evaluation heuristics, I perform this experiment in a deterministic version of the Marble Maze—actions have a 0 probability of failing. I test UCT agents with planning horizons 1 through 15, $N = 100$, and $c = 1$. All other experiment conditions were identical to the Sparse Sampling experiment.

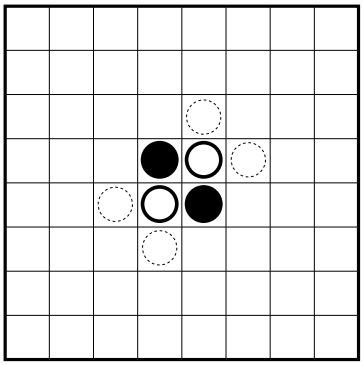


Figure 6.4: Othello game board, opening turn. Solid black and white circles are game pieces. Dashed circles represent legal opening moves for the black player.

Figure 6.3 presents the results for the UCT agents with horizons 5 and 15. Each learning curve plots the average of 50 trials. Predictably, increasing the planning horizon improves performance for agents which use the objective reward function, as shown by the fact that the square marker is above the diamond marker at 0 time steps. Also as predicted, the ability of a leaf-evaluation heuristic to improve performance is reduced as the tree is deepened, as shown by the superior performance of the Horizon-5 Leaf-Value(PGRD) agent (dashed line, diamond markers) compared to the Horizon-15 Leaf-Value(PGRD) agent (dashed line, square markers) and the similar trend observed in the V^*-leaf-evaluation heuristic agents (inset on right).

In contrast, general reward functions are more resilient to this bias at large planning horizons. The learned reward functions (top 2 curves) converge to near-optimal behavior at all tested horizons. Also, observe that the Horizon-5 Leaf-Value(PGRD) agent outperforms the corresponding Horizon-5 V^*-LEH agent shown in the inset, verifying the claim that an approximately optimal LEH can outperform the V^*-LEH when the agent is unable to exactly evaluate the expected return of the truncated trajectory.

6.5.2 Othello Experiment

Othello, also known as Reversi, is a two-player game with a long history in AI. Advancements in the quality of artificial Othello agents have been aided by advancements in the quality of Othello's leaf-evaluation functions (Buro, 2003). Recent studies have examined reinforcement learning methods and methods for learning evaluation functions for UCT in particular (Lucas, 2006; Hingston and Masek, 2007). Hingston

and Masek (2007), for example, attempted to find the L^*-LEH using a genetic programming approach. In this experiment, I examine the claim that reward functions can outperform leaf-evaluation heuristics in the context of Othello, using features of Othello state that were used as LEH features in previous work (Lucas, 2006; Hingston and Masek, 2007).

Environment. Othello is a board game in which two players, black and white, alternate turns by placing a piece on a game board. The black player places black pieces and wins the game if the board has more black pieces than white ones at the end. The white player places white pieces. If the players have the same number of pieces at the end of the game, it ends in a draw. The opening position is pictured in Figure 6.4. Black moves first. To make a legal move, black must place its piece such that there exists at least one straight (vertical, horizontal, or diagonal), contiguous line of white pieces between the new piece and a black piece already on the board. After black makes a legal move, all of the white pieces in such contiguous lines are flipped to black. Figure 6.4 illustrates all of the legal moves for black from the opening position (dashed circles). If a player does not have a legal move, the turn passes to the other player. The game ends when the board fills up, or neither player has a legal move. The standard-sized 8×8 game board has 64 tiles and starts with 4 of them occupied. A typical game will last close to 60 turns. Othello has an estimated 10^{28} states, an estimated game-tree complexity of 10^{58}, and is PSPACE-complete (in the $n \times n$ case) (Iwata and Kasai, 1994; Allis, 1994).

Objective Reward. In this experiment, I focus on a designer with the objective of building a black agent which can play strongly against a fixed white opponent, and I vary the fixed opponent across experiments. I encode the winning conditions of the game with an objective reward function that provides an objective reward of 1 if black wins the game, an objective reward of 0.5 if the game ends in a draw (a tie game), and an objective reward of 0 in the case of a loss or if the game is not yet over. The objective return function is the undiscounted sum of objective reward received during the game.

Reward Function Design Spaces. As in previous experiments, I compare two reward spaces: one is a subset of potential-based reward functions and the other is not. I first introduce the potential-based reward space. Recall that the potential-based reward space is $R_L(s, a, s'; \beta) = R_O(s, a, s') - L(s; \beta) + L(s'; \beta)$ for some $L(s; \beta)$

(given that $\gamma = 1$ for this objective return function). Othello is too large to use a representation for L which is capable of representing all functions of state. Instead, a compact representation must be used. I represent L as $L(s;\beta) = \beta^\mathsf{T}\phi_{\text{WPC}}(s)$ for some vector of parameters β, where the feature vector $\phi_{\text{WPC}}(s)$ is the Weighted Piece Counter (WPC) feature representation, a feature set used in previous work on learning evaluation functions in Othello (Lucas, 2006; Hingston and Masek, 2007). Specifically, each location on the board has one feature associated with it, for a total of 64 features. Each feature is $+1$, 0, or -1, if the corresponding location has a black piece, no piece, or a white piece, respectively.

In order to demonstrate that the reward function's benefit is due to its path-dependent effect on the agent's return estimates, and not on having a more expressive feature representation, I use the same feature representation for the path-dependent reward space. Specifically, I define the non-potential-based reward function space to be $R(s,a,s';\beta) = R_O(s,a,s') + \beta^\mathsf{T}\phi_{\text{WPC}}(s')$. I refer to this as the *after-state* reward representation, because it applies the WPC feature representation to the resulting state. Notice that the after-state reward representation is similar to the above potential-based reward representation, except it does not subtract the WPC features as a function of the start state s, removing the telescoping sum that was shown in Lemma 6.1. This small difference allows for path-dependent return modifications.

UCT Agent. Although the white agent is fixed, I do not assume that the UCT agent has a model of the opponent. When building a planning tree, however, the UCT algorithm must model the opponent's behavior somehow.

The UCT algorithm can be extended to plan in two-player zero-sum games by assuming the opponent plays a Nash equilibrium strategy (Gelly and Silver, 2008). Both black nodes and white nodes are in the planning tree, and the algorithm selects search-control actions for the black nodes as normal. For the white nodes, rather than selecting the action that maximizes the upper-confidence bound, the algorithm selects the search-control action that minimizes the lower-confidence bound. Specifically, the agent selects, $a^* = \arg\min_a Q(s,a,d) - c\sqrt{\frac{\log n(s,d)}{n(s,a,d)}}$ at a white node, where Q and n are defined as in Section 6.3.3. Thus, there are two types of tree nodes: min-nodes and max-nodes. Both use the same reward function. The difference is whether they select search-control actions to maximize or minimize the return.

In the experiments, both the black and white agents plan using this UCT algorithm. When executing its action in the world, the black agent selects it action using the softmax parameterized by the Q values at the root, as before. The white agent executes

actions according to a "soft-min" distribution $\mu_{\text{white}}(a|s;Q) \stackrel{\text{def}}{=} \frac{e^{-\tau Q(s,a)}}{\sum_b e^{-\tau Q(s,b)}}$ at the root.

I make one more modification to UCT that is commonly used in deployed UCT implementations (Gelly and Silver, 2008). The Q-estimates and counts n are often stored in a tree data structure which is constructed as nodes are sampled. When nodes are experienced for the first time, they are added to the tree. To handle the large scale of Othello, the implementation here adds only up to 1 new node to the end of the tree data structure per trajectory (effectively keeping the counts n and Q-estimates for other potential new nodes at 0.). Note that the agent's return estimates are still based on the entire length-H trajectory, and the leaf state—as far as the LEH is concerned—is the final state in the trajectory.

I use UCT parameters $N = 100$ and $c = 1$ and the softmax parameter $\tau = 100$.

PGRD-UCT in an Episodic Environment. Othello is an episodic task, rather than a continuing one. However, the OLPOMDP method, upon which PGRD is based, was designed for continuing tasks. Thus, I replace the OLPOMDP gradient update in PGRD with Williams' REINFORCE method (Williams, 1992). This amounts to (1) setting the gradient estimate z_t to 0 at the beginning of every episode, (2) waiting until the end of the episode to add the gradient estimate (multiplied by the learning rate) $\alpha_t z_t$ to the reward parameters, and (3) setting the ξ parameter to 1.

Experiment Set-up Because this is a two-player game, each experiment measures the performance of a black agent against a white agent. I fix the white agent as an agent which uses the objective reward function, and optimize the reward function of the black agent. This is a single-agent design setting, because I am only optimizing one agent, and the other agent is fixed. Specifically, in each experiment, I use PGRD-UCT to optimize a reward function for the black agent while playing against a fixed white agent that plans with UCT and the objective reward function. PGRD-UCT is used to design reward functions of the after-state and of the potential-based forms. I optimize over learning rates in the set $\{10^{-8}, 3 \times 10^{-8}, 10^{-7}, 3 \times 10^{-7}, 10^{-6}, 3 \times 10^{-6}\}$. I use PGRD parameter $\lambda = 1$.

As in the Marble Maze experiments, I additionally use SARSA(λ) to optimize a potential-based reward function for the black agent. I optimize over learning rates in the set $\{10^{-5}, 3 \times 10^{-5}, 10^{-4}, 3 \times 10^{-4}, 10^{-3}\}$ and use $\epsilon = 0$ and $\lambda = 0$.

It is infeasible to calculate the V^* LEH in Othello. As an alternative, I calculate an additional baseline LEH using self-play, a method that is popular for learning value functions and leaf-evaluation functions in two player games (Tesauro, 1995; Gelly and

Silver, 2008). Here, the self-play agent is a SARSA(0) agent. A self-play agent plays as both the black and white players. At each step, the agent selects actions ϵ-greedily with respect to its Q-function. If it is black's turn, it maximizes, otherwise it minimizes. It computes the Q-values using the SARSA(0) learning update rule after each turn. The WPC feature set is used for the Q-function representation in the SARSA(0) agent for self-play-learned LEH. Specifically, $Q(s, a; \theta) = R_O(s, a, s') + \theta^{\mathsf{T}} \phi_{\text{WPC}}(s')$ for some parameters θ. The θ is learned from 1,000,000 episodes of self-play with an exploration rate $\epsilon = 0.1$ and a learning rate of $\alpha = 0.001$. After the parameters are learned, the self-play potential-based reward function (leaf-evaluation heuristic) is represented as $R_L(s, a, s'; \theta)$. Specifically, I use the θ learned from self-play as the parameters in the potential-based reward function.

I test the effect of different choices of horizon by varying the horizon in *both* agents across three different experiments. In the $H = 2$ experiment, I test horizon-2 black agents vs. a horizon-2 white agent which uses the objective reward function. A horizon-2 agent accounts for its own next move as well as the opponent's next. An agent which uses the objective reward function will act randomly if it is more than 2 steps away from the objective. In the $H = 16$ experiment, I test horizon-16 black agents vs. a horizon-16 white agent which uses the objective reward function. These agents account for about 1/4 the length of a full game in their plans. Finally, in the $H = 32$ experiment, I test corresponding pairs of horizon-32 agents. This corresponds to about half the length of a full game.

Results

Horizon-2 versus Horizon-2. Figure 6.5 presents the results from the Horizon-2 Othello Experiment. Each curve is an average of 30 trials. The leftmost points represent the agent prior to learning; it is the performance of the black agent which uses the objective reward function vs. the white agent which uses the objective reward function. These agents average close to 0.5 objective reward per game, which is equivalent to black winning half of the games or tying every game. At the end of each learning curve, I plot the point with a 95% confidence interval. The variance is small enough that the intervals are obscured by the markers in the plot. To the right of the curves, I plot the performance of the leaf-value function generated from self-play. These points are the performance of the horizon-2 black agent with the self-play LEH against the horizon-2 white agent. Consistent with the Marble Maze experiments, which demonstrated that leaf-evaluation heuristics perform comparatively

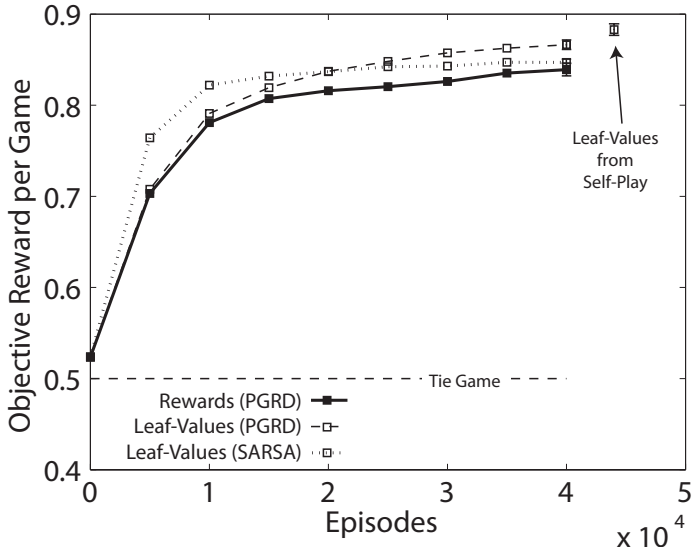


Figure 6.5: Results for the Horizon-2 Othello Experiment. Each curve presents a learning horizon-2 black agent versus a fixed horizon-2 white agent that uses the objective reward function.

well when the horizon is short, the after-state reward functions found (filled markers) do not outperform the leaf-evaluation heuristics found (open markers). Because the trajectories are short, the path-dependent nature of reward functions is not able to provide a benefit over leaf-evaluation heuristics.

Horizon-16 versus Horizon-16. Figure 6.6 presents results for the Horizon-16 Othello Experiment. At this horizon, the search control bias degrades UCT's return estimates, hampering the leaf-evaluation function's ability to improve performance. The approximately optimal reward function, in contrast, is more robust to this bias and outperforms all leaf-evaluation heuristic baselines.

Because the Horizon-16 experiment and the Horizon-2 experiments take place against different opponents, the curves in Figure 6.6 are not directly comparable to the curves in Figure 6.5. Notably, although the $H = 2$ black agents perform better against the $H = 2$ white agent than the $H = 16$ black agents perform against the $H = 16$ white agent, it is still the case that the $H = 16$ after-state-reward function agent is better than the $H = 2$ black agents. Specifically, after learning, the $H = 16$ after-state-reward function agent achieves expected objective reward of approximately 0.94 against the $H = 2$ white agent (out of a max of 1), whereas the $H = 2$ self-play LEH agent achieves only 0.88.

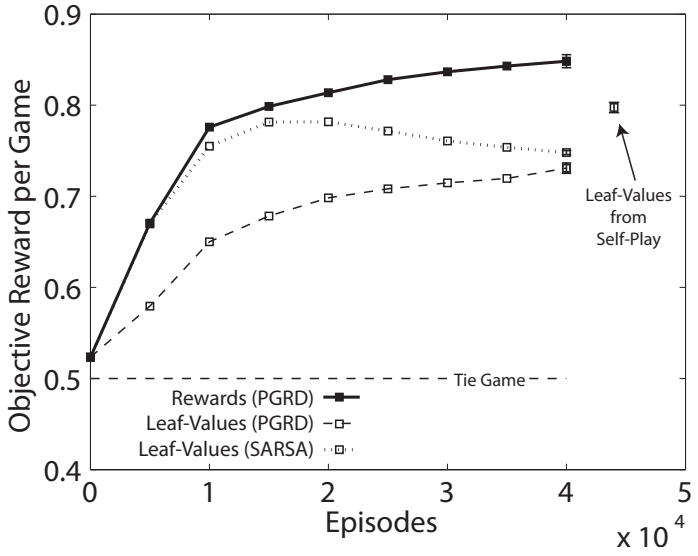


Figure 6.6: Results for the Horizon-16 Othello Experiment. Each curve presents a learning horizon-16 black agent versus a fixed horizon-16 white agent that uses the objective reward function.

Perhaps surprisingly, the performance of the SARSA-LEH agent (dotted line) degrades after initially improving. I hypothesize that this is because the SARSA agent is attempting to learn the V^* LEH, which ignores the agent architecture and is not guaranteed to be optimal. According to this explanation, the SARSA agent finds a LEH that works well early by chance, but performance degrades because SARSA is not selecting the parameters β based directly on their performance. The self-play LEH, in contrast, outperforms the horizon-16 SARSA LEH. The self-play LEH is learned in a different game (between two non-planning SARSA agents), thus it has different properties than the horizon-16 SARSA LEH. In this case it happens to outperform the horizon-16 SARSA LEH, but this is not guaranteed.

Horizon-32 versus Horizon-32. Figure 6.7 presents the results of the Horizon-32 Othello Experiment. Here, the optimal reward function approach (filled markers) outperforms the LEH approaches for the end-points of the trajectories. However, this only is because the performance of the SARSA-LEH agent again degrades after 1.5×10^4 games. The best reward function found (the end-point of the solid line) is not statistically different from the best LEH found (the dotted curve at 1.5×10^4).

Although reward functions are more resilient to search control bias, they are not immune. At horizon-32, the trajectories are very long relative to the $N = 100$

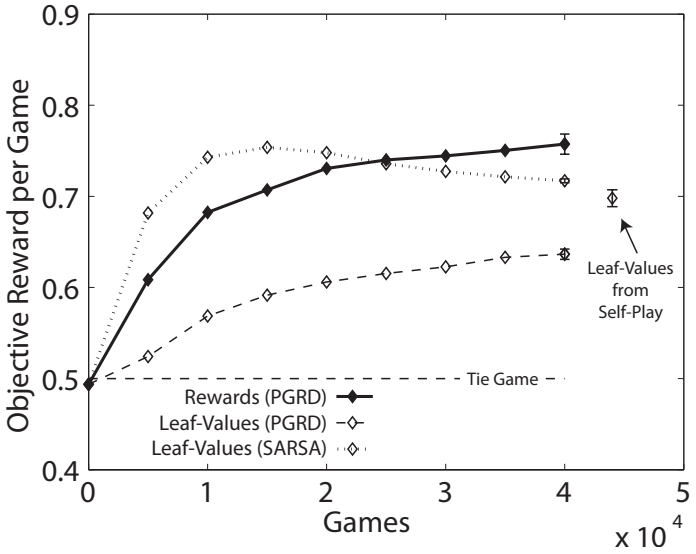


Figure 6.7: Results for the Horizon-32 Othello Experiment. Each curve presents a learning horizon-32 black agent versus a fixed horizon-32 white agent that uses the objective reward function.

trajectory count. Thus, estimates of the return from the sample trajectories will be very biased for both reward functions and leaf-value evaluation functions. This is one potential reason for why the horizon-32 experiment failed to differentiate leaf-evaluation heuristics from reward functions. I would expect the difference to reappear if the trajectory count N is increased.

Alternatively, it may differentiate the methods if instead of the WPC representation, a more expressive feature representation were used. This would allow the after-state representation to become "more path dependent." Using a more expressive feature representation would help leaf-evaluation heuristics as well, but I expect that it would help the path-dependent reward representation more.

Although the horizon-32 experiment fails to support the claim that an optimal reward function R^* is better than an optimal leaf evaluation heuristic L^*, this experiment does, however, support the claim that good optimization algorithms for leaf-evaluation heuristics and reward functions should take into account the limitations inherent in the agent. PGRD-UCT takes the agent's planning algorithm into account while optimizing, and its performance steadily improves. SARSA, however, ignores the agent architecture in its criterion for the LEH parameters. I conjecture that this leads to its degradation in performance over time.

Othello Summary

The horizon-2 experiment results were expected; the path-dependent nature of the after-state reward function is of little benefit when the trajectories are short relative to the sample count N, because there is little search-control bias. The horizon-16 experiment demonstrated that there exists parameter settings for UCT where the path-dependent nature of the after-state reward function is beneficial. At horizon-32, the benefit of the after-state reward representation over potential-based reward representation disappears. I hypothesize that this is because the search control bias is so strong in this experiment that even the path-dependent after-state representation is unable to overcome it.

PGRD-UCT successfully improved the performance of reward functions of both the after-state form and the potential-based form in all the experiments, confirming that PGRD-UCT can be applied in environments too large for current computational resources to find the optimal policy.

6.6 Discussion

In large-scale local planning algorithms such as UCT, increasing the planning horizon reduces the finite horizon bias, but it comes with other costs. First, it costs additional computational resources. Second, it enlarges the tree, increasing the potential for additional errors such as the sample variance and the search-control bias to affect value calculations and to impact the ability of leaf-evaluation heuristics to help.

I have demonstrated that due to the added expressiveness of path dependence, the optimal reward function approach is better than the leaf-evaluation heuristic approach and the potential-based reward shaping approach at improving the performance of planning agents which make approximations in shallow portions of the planning tree. Specifically, I showed that reward design is preferred in agents affected by sampling errors and search-control bias. Although additional heuristics exist for overcoming some of the agent limitations presented here and for specific algorithms (Gelly and Silver, 2008), solving the optimal reward problem is a general, principled approach which applies across agent limitations and algorithms.

I presented convergent and approximate online reward function optimization methods for Sparse Sampling and UCT, respectively, and used these methods to learn approximately optimal reward functions and leaf-evaluation heuristics for these methods. I demonstrated that these methods are effective at improving agent performance

in two environments: Marble Maze and Othello. The Othello experiments give hope that the reward design approach can scale to even larger environments. Future work will examine reward design in a full multi-agent setting.

Chapter 7
Discussion and Future Work

In the first half of this concluding chapter, Section 7.1, I summarize the results of this dissertation. This dissertation focuses on the optimal reward problem from the perspective of an agent designer who is already committed to designing an RL agent—one that attempts to maximize its expected return. This perspective is important because it provides advice to RL agent designers for how to improve their agents: design the agent's reward function.

In the second half of this chapter, I address a more fundamental question: in the context of bounded agent design, why use an RL agent? In other words, this dissertation advocates that when a reward-based agent cannot effectively maximize the objective return, the designer should build an agent that does not attempt to do so. There are two alternative ways to build an agent which does not attempt to maximize the objective return: (1) choose among agents that optimize return functions, but give the agent its own return function (the approach taken here), or (2) drop the notion of return maximization entirely and choose among agents from the set of all agents, not just those that attempt to maximize return functions. In section 7.2. I provide arguments for why approach (1) may be better. In particular, I show that the optimal reward problem is as general as the generic agent optimization problem, and I argue that the additional structure imposed by the optimal reward problem can be helpful to agent designers, because reward optimization has a number of appealing properties. I conclude by discussing open problems and future work in Section 7.3.

7.1 Summary of this Dissertation

The central claim of this dissertation, which I have repeatedly demonstrated through experiment, is that solving the optimal reward problem is advantageous when an agent is limited. However, experiments in Chapters 3 and 5 confirm that when an agent is

not limited, the objective reward function is sufficient.

Designing reward is a challenging problem. A limited agent, by definition, does not act optimally. Therefore, its behavior is not easily characterized by examining properties of the reward function or the environment alone. It is critical to understand the *nature of the agent's limitations* and how they will interact with the environment. This will help guide an agent designer's choice of the reward function. For example, in the Foraging Environment experiments in Chapters 3 and 5, the agent could not plan accurately or far enough into the future to locate a moving food source, either due to a limited state representation or limited planning horizon. These limitations prevented it from accurately locating the food. I provided the agent with the inverse-recency feature, which motivated the agent to explore, putting the food source within range of its accurate planning. In the Dark Room environment in Section 3.2.3, I provided the agent with the model-inaccuracy feature, which motivated the agent to avoid locations it had trouble modeling, preventing it from wasting time executing low-value actions.

The Dark Room experiment is an example where, if the agent's limitations are severe enough, the optimal reward function can motivate behavior that does not resemble optimal behavior. There, the optimal reward function motivated the agent to take the long path around the dark room, even though optimal behavior would travel through the room. In the Bait-or-Fish experiment (Section 2.2), the optimal reward function for some experiment horizons chooses to motivate eating bait—the behavior that is easier to learn but provides less objective reward.

In Chapter 4, I derive the variance-based reward bonus, a reward feature which approximates Bayes-optimal exploration. The agent maintains knowledge about the world in the form of a Bayesian posterior distribution, but does not have enough resources to perform full Bayesian planning. Thus, it settles for planning in the Mean MDP. Due to the Mean MDP approximation, the agent cannot account for the value of gaining information from exploring unknown locations. The variance-based reward bonus, which is a measure of how much information the agent would learn by sampling each action, produces provably efficient exploration in the form of a bound on the number of samples required to achieve near-optimal performance with high probability (Theorem 4.1). In the Wumpus World experiment in Section 4.3.2, I demonstrated that the variance-reward bonus is better than alternative approximate Bayesian approaches at exploiting structured knowledge in the prior distribution. One reason for this is that the additive nature of rewards properly weighs opposing motives (explore vs. exploit). This is opposed to the BOSS algorithm, for example, which does not have a parameter for explicitly weighing the explore and exploit alternatives.

While reward features such as the ones above are helpful in selecting the reward design space $\mathcal{R}$, finding the optimal reward function $R^* \in \mathcal{R}$ still requires fine-tuning. For example, in the Wumpus World experiment, how much should the agent value exploiting (objective reward) vs. exploring (variance-based bonus)? Current theory provides limited insight. Theorem 4.1, for example, provides theoretical guarantees bounding the agent's learning sample complexity as a function of the reward parameters; however, these worst-case bounds don't provide an analytical solution to the ORP. Due to the complex interaction between the agent's limitations and the environment, solving the ORP in general may ultimately require empirical solutions.

To address this, I propose a novel reward design algorithm, Policy Gradient for Reward Design. PGRD and its variants are algorithms which approximately solve the ORP empirically during an agent's lifetime by viewing the reward parameter optimization problem as a policy parameter optimization problem. Solving the ORP during an agent's lifetime is challenging because an agent's limitations interact with the environment in complex ways; however, policy gradient methods are well suited for dealing with this complexity because they are designed for finding solutions to partially observable systems without having to model their dynamics. PGRD was originally developed in Chapter 5 for the FFHP algorithm and was extended in Chapter 6 for the Sparse Sampling and UCT algorithms.

As an additional contribution of Chapter 6, I discussed how UCT suffers from multiple limitations. I demonstrated that an evaluation function at the leaf node of the planning tree can help UCT overcome the finite-horizon bias, but a reward function can be better at overcoming the search control bias and sample variance, while simultaneously overcoming the finite-horizon bias. I also demonstrated that potential-based shaping reward functions and leaf-evaluation functions are equivalent. Thus, as a corollary of this demonstration, it was shown that potential-based shaping reward functions are not always optimal. I demonstrated these final claims in two environments, including Othello, a two-player game with 10^{28} states.

7.2 The General Agent-Design Problem

In this section, I examine the optimal reward problem in the context of the broader bounded-agent design process. In their work "Provably Bounded-Optimal Agents," Russell and Subramanian (1995) provided a definition of bounded optimality that does not involve the optimal reward problem. Let $\tilde{\Theta}$ be the bounded set of agent

parameters which are computationally feasible. Recall that an agent G is a mapping from histories to distributions over actions. Therefore, the set $\tilde{\Theta}$ corresponds to the set of such mappings which can be feasibly computed online as experience is received, given the agent's limited computational resources[1]. The *conventional bounded-optimal agent* is given by the solution to the conventional bounded-agent optimization problem: $\tilde{\theta}^* = \arg\max_{\tilde{\theta} \in \tilde{\Theta}} \mathbb{E}\left[U_{R_O}(h) | h \sim M\langle G(\tilde{\theta})\rangle\right]$. Agent $G(\tilde{\theta}^*)$ is the best agent a designer can build *given the computational constraints*.

In Chapter 3, in the table of four related agent design spaces (Table 3.1), I defined an alternative definition of a bounded-optimal agent which considers the reward function to be an additional parameter to the agent. Recall that this agent is the best agent from the joint design space $(\mathcal{R} \times \Theta)$, where $\mathcal{R}$ is the set of reward functions and Θ is a bounded set of agent parameters. Specifically, Θ is a bounded set of mappings from histories to distributions over actions, parameterized by reward function.

The set Θ I defined in Chapter 3 is bounded differently from the set $\tilde{\Theta}$ above. Specifically, while $\tilde{\Theta}$ is constrained only by computational resources, I allow Θ to be constrained by additional design constraints. Most important among these, I allow the designer to constrain Θ to contain only agent designs θ which seek to maximize their expected return, as defined by the reward function parameter. To more directly distinguish it from the above bounded-optimality definition, I refer to the definition in Chapter 3 as a *reward-based bounded-optimal agent*. A reward-based bounded-optimal agent $G(R^*, \theta^*)$ is the best an agent designer could build *given the design constraints*. I refer to the corresponding optimization problem as the reward-based bounded-agent optimization problem. The definition in Chapter 3 does more than simply add an additional constraint; it adds structure to the problem by breaking it up into an optimization over conventional agent parameters and reward parameters.

It should be clear that because the conventional bounded-optimality condition defines a broader space of agents, the conventional bounded-optimal agent will perform at least as well as the reward-based bounded-optimal agent. However, I argue that the reward-based bounded-optimal agent is a good agent because (1) the set of agents in the reward-based bounded-agent optimization problem is not limited compared to the set of agents in the conventional bounded-agent optimization problem, and (2) the reward function search space has desirable properties, some of which I touch on below.

[1]Russell and Subramanian (1995) describe $\tilde{\Theta}$ as the set of all agent *programs* which are feasible on a given machine.

7.2.1 The ORP is a Fully-General Method of Agent Design

Recall Theorem 5.3. That theorem showed that there exists a reward function space $\mathcal{R}$ which is capable of causing an FFHP agent which selects its actions using a softmax distribution at the root node to act according to any stochastic policy, regardless of the transition function. Theorem 5.3 was proven for policies over (assumed) Markov state, but it could easily be extended to be shown for policies that depend arbitrarily on history.

Thus, for every mapping from histories to distributions over actions, there exists a reward function that will cause an FFHP agent with softmax action selection to choose actions according to that mapping. In other words, with an appropriate choice of a fixed θ and reward design space $\mathcal{R}$, solving the optimal reward problem (solving the reward-based bounded-optimality problem) is equivalent to solving the conventional bounded-optimality problem.

7.2.2 The Reward Design Space has Desirable Properties

The reward design space has a number of desirable properties that distinguish it from conventional agent design parameters. I highlight some of these below.

Choosing *what* to do or *how* to do it. It was stated in the introduction that reward functions allow a designer to specify what do to (the goal) without having to tell the agent how to do it (the policy). In fact, agent designers can specify both through the reward function: reward design allows a designer to specify what to do and/or how to do it, providing agent designers with a trade-off in their agent specification. They can choose to specify their agent in terms of abstract goals, in terms of direct commands, or a broad space of objectives in between. In this section, I present an intuitive argument to support this claim.

Most RL algorithms take into account the transition dynamics when converting the reward function R into behavior. However, there exists some reward functions such that the optimal policy can be found without considering the environment dynamics. Given a deterministic policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, define the policy-specific reward function:

$$R_\pi(s, a) = 0 \quad \text{if} \quad \pi(s) = a$$
$$R_\pi(s, a) = -1 \quad \text{if} \quad \pi(s) \neq a.$$

It can be easily shown that the unique optimal policy given R_π is π, regardless of the

agent's transition model. In FFHP (with greedy action selection), for example, the agent will act according to π starting from time step 0 for any planning horizon $H \geq 1$ regardless of how the transition model is learned. This result can easily be extended to define arbitrary policies over the agent's history. Thus, reward functions can be used to directly specify complex learning or exploratory behavior.

In contrast, consider a goal-based reward function: $R_G(s, a) = 1$ if the agent is at the goal state G and is 0 otherwise. It can easily be shown that any policy is potentially optimal when using this reward function, depending on the transition function. Thus, the agent's planning and model-learning algorithms—and therefore, limitations—will more greatly affect the agent's behavior.

Reward design allows a designer to flexibly specify reward functions between these two extremes. The less limited an agent is, the better it is able to perform when given reward functions that share characteristics with the goal-based reward function above. Combined with the generality of the reward-based bounded-agent optimization problem observed in the previous section, reward design allows a designer to attempt to solve the *conventional* bounded-agent optimization problem while taking advantage of this flexibility.

Reward functions generalize well across environments. I argue that designed reward functions and reward features are likely to generalize well across environments. This is because the reward function affects the policy through the transition model, which differs across environments. Therefore, a reward function's effect on the agent's behavior will automatically adapt to the specifics of the environment. Value-function-based heuristics or policy-based heuristics, in contrast, don't automatically account for differences in transition models across environments. The trade-off, of course, is that reward functions can only affect behavior through expensive processes such as planning or learning, but policy-based heuristics can be applied immediately.

Experiments 3 and 4 in Chapter 3 support this conjecture; the same reward function was used in in two separate environments and achieved near-optimal performance, but the objective reward function performed poorly in both.

7.3 Future Work

This dissertation made a few simplifying assumptions that eased analysis which future work should examine. Also, there are a number of current challenges which could be

mitigated by future insight. I outline a few areas for future work below.

Costs of Reward Design

This dissertation demonstrated that solving the optimal reward problem is beneficial if one ignores the expense of finding the solution. Future work should perform a cost/benefit analysis of optimizing the reward function parameters and compare it to spending similar amounts of computation in other places such as: (1) in the agent itself or (2) on optimizing the agent's conventional parameters.

There is hope that such an analysis will yield favorable results for reward design. PGRD, for example, is only linear in the number of reward function parameters, meaning the computational complexity it adds to the agent is not large.

This dissertation also ignored the computational cost of computing the reward function itself. This is not trivial, as reward functions which are complicated functions of history could require large amounts of memory. The above generality results on reward design show that it is possible to remove all computation from the agent and put it into the reward function. Given that the reward function is typically implemented on the same machine as the agent (Chapter 1), future work should account for this cost.

Joint Design of the Reward-Function and Conventional Agent Parameters

Although it was discussed in this chapter and in Chapter 3, this dissertation did not generally focus on jointly optimizing the reward function and conventional agent parameters. It may be the case, for example, that it is easier to solve the joint problem than it is to solve either problem by itself. For example, FFHP required minor modifications (softmax action selection) to make reward optimization possible in PGRD. Also, perhaps solving the joint problem will result in agent designs which take advantage of specific reward function properties.

Multi-Agent Reward Design

The Othello experiment illustrated that there are instances where designers could benefit from studying the reward design problem in the multi-agent setting. In the Othello experiment, I avoided the issue of multi-agent reward design by fixing the opponent; thus, it could be modeled as a single agent problem.

Optimal reward functions in this setting would likely follow game-theoretic principles (Samuelson, 2001). Not only does theory need to be extended to account for multiple agents, new algorithms will likely need to be developed. For example, in two-player zero-sum games, self play has been shown to work well for learning value functions Tesauro (1995); Gelly and Silver (2008), but it is not clear whether a policy gradient method like PGRD will successfully optimize reward functions in self-play.

Nested Reward Design Problem

PGRD does not explore the environment in a principled manner. It simply follows its current policy (plans with its current reward function) and uses the experience that results to try and improve the reward parameters. It typically starts with a random policy or the policy defined by the objective reward, which is random when the goal states are beyond the horizon. Thus, exploration in PGRD is effectively random. Compare that to algorithms like E^3 (Kearns and Singh, 2002), RMax (Brafman and Tennenholtz, 2001), or the variance-based reward method in Chapter 4, which have systematic exploration behavior. How would one achieve this in PGRD? Notice that these explicit exploration methods use reward design to encourage exploration. Notice also that PGRD is, itself, an agent which maximizes the objective reward (to improve the planning algorithm's reward function parameters). This brings up an interesting question: is it useful to design the reward function of a reward designing agent?

Finding Good Reward Design Spaces

A negative consequence of the fact that the reward design space is very flexible, as discussed above in Section 7.2.2, is that reward function search space is a very broad space. In this dissertation, I presented some reward features, such as the variance-based reward feature and the recency reward feature, which can be used across environments. The hope is that environment-independent reward features such as these can be used to aid in the selection of reward design spaces. Future work should provide guidance in the selection of the reward design space.

Provably Optimal Solutions to the ORP

Finally, although this dissertation has theory which lends insight into the optimal reward problem, the solutions here were found and evaluated empirically. For example, although the variance reward work shows that there exist reward functions with strong

theoretical guarantees, the theorems did not provide solutions to the ORP. Future work should seek instances of the optimal reward problem with analytically derivable solutions and efficient algorithms which are provably able to find (globally) optimal solutions.

Bibliography

Pieter Abbeel and Andrew Y. Ng. Apprenticeship Learning via Inverse Reinforcement Learning. *Twenty-first international conference on Machine learning - ICML '04*, 2004.

Douglas Aberdeen and Jonathan Baxter. Scalable Internal-State Policy-Gradient Methods for POMDPs. *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

R. Agrawal. Sample mean based index policies with O(log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, 1995.

Louis V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. Phd, College van Dekanen, 1994.

John Asmuth, Michael L. Littman, and Robert Zinkov. Potential-based Shaping in Model-based Reinforcement Learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI-08)*, pages 604–609. AAAI Press, 2008.

John Asmuth, Lihong Li, Michael L. Littman, Ali Nouri, and David Wingate. A Bayesian Sampling Approach to Exploration in Reinforcement Learning. In *Proceedings of the Twenty-Fifth Conference Annual Conference on Uncertainty in Artificial Intelligence*, pages 19–26, 2009.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2):235–256–256, 2002.

Peter L. Bartlett and Jonathan Baxter. Stochastic optimization of controlled partially observable markov decision processes. In *Proceedings of the 39th IEEE Conference on Decision and Control (CDC00)*, 2000.

Andrew G. Barto, Satinder Singh, and N Chentanez. Intrinsically Motivated Learning of Hierarchical Collections of Skills. In *Proceedings of the International Conference on Developmental Learning (ICDL)*, LaJolla, CA, 2004.

Jonathan Baxter, Peter L. Bartlett, and Lex Weaver. Experiments with Infinite-Horizon, Policy-Gradient Estimation. *Journal of Artificial Intelligence Research*, 15: 351–381, 2001.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

Shalabh Bhatnagar, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor-critic algorithms. *Automatica*, 2009.

Ronen I. Brafman and Moshe Tennenholtz. R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3:213–231, 2001.

M Buro. The Evolution of Strong Othello Programs. In *Entertainment Computing - Technology and Applications*, pages 81–88, 2003.

Michael Duff. *Optimal Learning: Computational Procedures for Bayes-Adaptive Markov Decision Processes*. PhD thesis, University of Massassachusetts Amherst, 2002.

Michael Duff. Design for an Optimal Probe. In *Proceedings ot the Tentieth International Conference on Machine Learning*, 2003.

S. Elfwing, Eiji Uchibe, K. Doya, and H. I. Christensen. Co-evolution of Shaping Rewards and Meta-Parameters in Reinforcement Learning. *Adaptive Behavior*, 16 (6):400–412, 2008.

Hilmar Finnsson and Y. Björnsson. Simulation-based approach to general game playing. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI*, pages 259–264, 2008.

Sylvain Gelly and David Silver. Achieving Master Level Play in 9 x 9 Computer Go. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, 2008.

Marek Grześ and Daniel Kudenko. *Multigrid Reinforcement Learning with Reward Shaping*, volume 5163 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

Marek Grześ and Daniel Kudenko. Learning Shaping Rewards in Model-based Reinforcement Learning. In *Proc. of AAMAS 2009 Workshop on Adaptive Learning Agents (ALA 2009)*, 2009.

Vijaykumar Gullapalli and Andrew G. Barto. Shaping as a Method for Accelerating Reinforcement Learning. In *Proceedings of the IEEE International Symposium on Intelligent Control*, pages 554–559, 1992.

P. Hingston and M. Masek. Experiments with Monte Carlo Othello. In *2007 IEEE Congress on Evolutionary Computation*, pages 4059–4064. Evolutionary Computation, September 2007.

Shigeki Iwata and Takumi Kasai. The Othello game on an n n board is PSPACE-complete. *Theoretical Computer Science*, 123(2):329–340, January 1994.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

S. Kakade, Michael Kearns, and J. Langford. Exploration in metric state spaces. In *Proceedings of the 20th International Conference on Machine Learning*, page 306, 2003.

Michael Kearns and Satinder Singh. Near-Optimal Reinforcement Learning in Polynomial Time. In *Machine Learning*, pages 260–268. Morgan Kaufmann, 1998.

Michael Kearns and Satinder Singh. Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning*, 49(2), 2002.

Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1324–1331, 1999.

Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning*, pages 282–293, 2006.

J. Zico Kolter and Andrew Y. Ng. Near-Bayesian Exploration in Polynomial Time. In *Proceedings of the 26th International Conference on Machine Learning*, pages 513–520, 2009.

George Konidaris and Andrew G. Barto. Autonomous Shaping: Knowledge Transfer in Reinforcement Learning. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 489–496, New York, 2006.

Harold J. Kushner and G. George Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer, 2nd edition, 2010.

A.D. Laud. *Theory and application of reward shaping in reinforcement learning*. PhD thesis, University of Illinois at Urbana-Champaign, 2004.

Simon M. Lucas. Temporal Difference Learning Versus CoEvolution for Acquiring Othello Position Evaluation. In *IEEE Symposium on Computational Intelligence and Games*. IEEE Symposium on Computational Intelligence and Games, 2006.

S Mahadevan. To discount or not to discount in reinforcement learning: A case study comparing R learning and Q learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, 1994.

Bhaskara Marthi. Automatic shaping and decomposition of reward functions. *Proceedings of the 24th international Conference on Machine Learning*, pages 601–608, 2007.

Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, USA, 1995.

Maja J Mataric. Reward Functions for Accelerated Learning. *Proceedings of the 11th International Conference on Machine Learning*, 1994.

Çetin Meriçli, Tekin Meriçli, and H. Levent Akin. A Reward Function Generation Method Using Genetic Algorithms : A Robot Soccer Case Study (Extended Abstract). In *Proc. of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1513–1514, 2010.

Gergely Neu and Csaba Szepesvári. Apprenticeship Learning using Inverse Reinforcement Learning and Gradient Methods. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, pages 295–302, 2007.

Andrew Y. Ng and Stuart J. Russell. Algorithms for Inverse Reinforcement Learning. In *Proceedings of the 17th International Conference on Machine Learning*, 2000.

Andrew Y. Ng, Stuart J. Russell, and D. Harada. Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pages 278–287, 1999.

Scott Niekum, Andrew G. Barto, and Lee Spector. Genetic Programming for Reward Function Search. *IEEE Transactions on Autonomous Mental Development*, 2(2): 83–90, 2010.

Pierre-Yves Oudeyer and Frederic Kaplan. What is Intrinsic Motivation? A Typology of Computational Approaches. *Frontiers in Neurorobotics*, 2007.

Pierre-Yves Oudeyer, Frederic Kaplan, and Verena V. Hafner. Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, April 2007.

Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An Analytic Solution to Discrete Bayesian Reinforcement Learning. In *Proceedings of the 23rd international conference on Machine learning*, 2006.

Deepak Ramachandran and Eyal Amir. Bayesian Inverse Reinforcement Learning. In *Proceedings of the 20th International Joint Conference on Artifical intelligence*, volume 51, pages 2586—-2591, Hyderabad, India, 2007.

Jette. Randløv and Preben. Alstrøm. Learning to Drive a Bicycle using Reinforcement Learning and Shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 463–471. Citeseer, 1998.

Kevin Regan and Craig Boutilier. Regret-based Reward Elicitation for Markov Decision Processes. In *The 25th Conference on Uncertainty in Artificial Intelligence*, 2009.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall; 2nd edition, 2002.

Stuart J. Russell and Devika Subramanian. Provably Bounded-Optimal Agents. *Journal of Artificial Intelligence Research*, 2:575–609, 1995.

R M Ryan and E L Deci. Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. *Contemporary Educational Psychology*, 25:54–67, 2000.

A. L. Samuel. Some Studies in Machine Learning using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

Larry Samuelson. Introduction to the Evolution of Preferences. *Journal of Economic Theory*, 97:225–230, 2001.

Larry Samuelson and Jeroen Swinkels. Information, evolution and utility. *Theoretical Economics*, 1(1):119–142, 2006.

Jürgen Schmidhuber. A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 222–227, Cambridge, MA, 1991a. MIT Press.

Jürgen Schmidhuber. Adaptive Confidence and Adaptive Curiosity. Technical Report FKI-149-91, Institut für Informatik, Technische Universität München, Arcisstr. 21, 800 München 2, Germany, 1991b.

Jürgen Schmidhuber. Curious model-building control systems. In *IEEE International Joint Conference on Neural Networks*, pages 1458–1463, 1991c.

Jürgen Schmidhuber. What's Interesting? Technical Report TR-35-97, IDSIA, Lugano, Switzerland, 1997.

Jürgen Schmidhuber. Artificial Curiosity Based on Discovering Novel Algorithmic Predictability Through Coevolution. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1612–1618. IEEE Press, 1999.

Jürgen Schmidhuber. Self-Motivated Development Through Rewards for Predictor Errors/Improvements. In *Developmental Robotics 2005 AAAI Spring Symposium*, pages 1994–1996, 2005.

Claude E. Shannon. XXII. Programming a computer for playing chess. *Philosophical Magazine (Series 7)*, 1950.

David Silver, Richard S. Sutton, and Martin Müller. Sample-Based Learning and Search with Permanent and Transient Memories. In *Proceedings of the 25th international conference on Machine Learning*, pages 968–975, New York, New York, USA, 2008. ACM Press.

Özgür Şimşek and Andrew G. Barto. An Intrinsic Reward Mechanism for Efficient Exploration. In *Proceedings of the 23rd International Conference on Machine learning*, pages 833–840, New York, New York, USA, 2006. ACM Press.

Satinder Singh, Andrew G. Barto, and Nuttapong Chentanez. Intrinsically Motivated Reinforcement Learning. In *Proceedings of Advances in Neural Information Processing Systems 17 (NIPS)*, pages 1281–1288, 2005.

Satinder Singh, Richard L. Lewis, and Andrew G. Barto. Where Do Rewards Come From? In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 2601–2606, 2009.

Satinder Singh, Richard L. Lewis, Andrew G. Barto, and Jonathan Sorg. Intrinsically Motivated Reinforcement Learning: An Evolutionary Perspective. *IEEE Transations on Autonomous Mental Development*, 2(2):70–82, 2010.

B. F. Skinner. *The Behavior of Organisms: An Experimental Analysis.* Prentice Hall, Englewood Cliffs, New Jersey, 1938.

Jonathan Sorg, Satinder Singh, and Richard L. Lewis. Internal Rewards Mitigate Agent Boundedness. In *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010a.

Jonathan Sorg, Satinder Singh, and Richard L. Lewis. Reward Design via Online Gradient Ascent. In *Advances of Neural Information Processing Systems 23*, 2010b.

Jonathan Sorg, Satinder Singh, and Richard L. Lewis. Variance-Based Rewards for Approximate Bayesian Reinforcement Learning. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence*, 2010c.

Jonathan Sorg, Satinder Singh, and Richard L. Lewis. Optimal Rewards versus Leaf-Evaluation Heuristics in Planning Agents. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI-11)*, 2011.

Alexander L. Strehl and Michael L. Littman. A Theoretical Analysis of Model-Based Interval Estimation. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML)*, pages 857–864. ACM Press, 2005.

Alexander L. Strehl and Michael L. Littman. An Analysis of Model-Based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

Alexander L. Strehl, L Li, Eric Wiewiora, and J Langford. PAC Model-Free Reinforcement Learning. In *Proceedings of the 23rd international conference on Machine learning (ICML)*, pages 881–888, 2006a.

Alexander L. Strehl, Lihong Li, and Michael L. Littman. Incremental Model-Based Learners with Formal Learning-Time Guarantees. In *Proceedings of the 22nd conference on Uncertainty in Artificial Intelligence*, pages 485–493, 2006b.

Alexander L. Strehl, Carlos Diuk, and Michael L. Littman. Efficient Structure Learning in Factored-State MDPs. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, 2007.

Richard S. Sutton. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *The Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990.

Richard S. Sutton. The reward hypothesis, 2004. URL `http://rlai.cs.ualberta.ca/RLAI/rewardhypothesis.html`.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112:181–211, 1999.

Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 1995.

Eiji Uchibe and Kenji Doya. Finding Intrinsic Rewards by Embodied Evolution and Constrained Reinforcement Learning. *Neural Networks*, 21(10):1447–1455, 2008.

Lex Weaver and Nigel Tao. The Optimal Reward Baseline for Gradient-Based Reinforcement Learning. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 538–545. Morgan Kaufman Publishers, 2001.

Eric Wiewiora. Potential-Based Shaping and Q-Value Initialization are Equivalent. *Journal of Artificial Intelligence Research*, 19, 2003.

Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3-4):229–256, May 1992.