AN ABSTRACT OF THE DISSERTATION OF

Aaron Wilson for the degree of Doctor of Philosophy in Computer Science presented on July 28, 2012.

Title: Bayesian Methods for Knowledge Transfer and Policy Search in Reinforcement Learning

Abstract approved: _____

Prasad Tadepalli Alan Fern

How can an agent generalize its knowledge to new circumstances? To learn effectively an agent acting in a sequential decision problem must make intelligent action selection choices based on its available knowledge. This dissertation focuses on Bayesian methods of representing learned knowledge and develops novel algorithms that exploit the represented knowledge when selecting actions.

Our first contribution introduces the multi-task Reinforcement Learning setting in which an agent solves a sequence of tasks. An agent equipped with knowledge of the relationship between tasks can transfer knowledge between them. We propose the transfer of two distinct types of knowledge: knowledge of domain models and knowledge of policies. To represent the transferable knowledge, we propose hierarchical Bayesian priors on domain models and policies respectively. To transfer domain model knowledge, we introduce a new algorithm for model-based Bayesian Reinforcement Learning in the multi-task setting which exploits the learned hierarchical Bayesian model to improve exploration in related tasks. To transfer policy knowledge, we introduce a new policy

search algorithm that accepts a policy prior as input and uses the prior to bias policy search. A specific implementation of this algorithm is developed that accepts a hierarchical policy prior. The algorithm learns the hierarchical structure and reuses components of the structure in related tasks.

Our second contribution addresses the basic problem of generalizing knowledge gained from previously-executed policies. Bayesian Optimization is a method of exploiting a prior model of an objective function to quickly identify the point maximizing the modeled objective. Successful use of Bayesian Optimization in Reinforcement Learning requires a model relating policies and their performance. Given such a model, Bayesian Optimization can be applied to search for an optimal policy. Early work using Bayesian Optimization in the Reinforcement Learning setting ignored the sequential nature of the underlying decision problem. The work presented in this thesis explicitly addresses this problem. We construct new Bayesian models that take advantage of sequence information to better generalize knowledge across policies. We empirically evaluate the value of this approach in a variety of Reinforcement Learning benchmark problems. Experiments show that our method significantly reduces the amount of exploration required to identify the optimal policy.

Our final contribution is a new framework for learning parametric policies from queries presented to an expert. In many domains it is difficult to provide expert demonstrations of desired policies. However, it may still be a simple matter for an expert to identify good and bad performance. To take advantage of this limited expert knowledge, our agent presents experts with pairs of demonstrations and asks which of the demonstrations best represents a latent target behavior. The goal is to use a small number of queries to elicit the latent behavior from the expert. We formulate a Bayesian model of the querying process, an inference procedure that estimates the posterior distribution

over the latent policy space, and an active procedure for selecting new queries for presentation to the expert. We show, in multiple domains, that the algorithm successfully learns the target policy and that the active learning strategy generally improves the speed of learning.

Bayesian Methods for Knowledge Transfer and Policy Search in Reinforcement Learning

by

Aaron Wilson

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented July 28, 2012
Commencement June 2013

Doctor of Philosophy dissertation of Aaron Wilson presented on July 28, 2012.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Aaron Wilson, Author

ACKNOWLEDGEMENTS

The faculty of Computer Science at Oregon State University were inspiring figures during my undergraduate and graduate career. They are standouts as both scientists and people and because of this I have been reluctant to leave this place. Certain members stand out prominently in my mind. Margaret Burnett gave me an opportunity to begin my research career as an undergraduate student. She is an enthusiastic pursuer of ideas and inspired me to continue on into graduate level research. Early in my studentship Tom Dietterich was kind enough to run a reading and conference to study Bayesian approaches to robot localization. His love of the material was obvious during our meetings and his willingness to answer questions from a new researcher highlighted the value of a curious spirit. Prasad Tadepalli has acted as a mentor throughout my studies. His interest in the "big questions" of artificial intelligence convinced me to stay at OSU to pursue my graduate studies. He has been invaluable in both encouraging and directing my curiosity and it has been a pleasure to work with him. Alan Fern provided insight and guidance during our many meetings. His efforts were invaluable and contributed, substantially, to the intellectual quality of this work. It has been a privilege to work here. Thank you.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF ALGORITHMS

To my father and my mother.

Chapter 1: Introduction

Animals possess minute computational resources by contrast to the world they live in. Despite their considerable limitations, animals learn a variety of cognitive functions through periods of extensive world exploration. Singular among animals is the protracted period of cognitive development experienced by human beings. The capabilities learned during human development are so extensive that synthesis to a coherent theory of cognitive function acquisition has remained beyond reach. Instead, communities studying the problem of learning have produced myriad illustrative examples of cognitive capabilities, and have indicated distinct explanations for acquisition. Within the machine learning community the story is much the same. Distinct problems receive new and distinct solutions. Though all solutions were produced by human brains, no theory explains their genesis. Because a synthesis remains outside our reach, this thesis will proceed by proposing the study of a particular problem: intelligent experimentation by embodied agents in sequential decision problems.

Sequential decision problems are pervasive in our life experience. Consider the familiar problem of preparing a dish for dinner. To do this it is necessary to follow a sequence of steps. Vegetables and meat need to be prepared, ingredients need to be located in pantries, stoves need to be preheated, and so on. Eventually, the prepared ingredients must be combined, sometimes in very specific ways, and the combined ingredients must be placed in the oven to cook. Errors along the way have consequences for dinner. If, while preparing bread, the leavening has been forgotten, then the bread will not rise. Our early decisions have *delayed consequences*. We may not know that the

leavening has been forgotten until we observe the final state of the bread, and experience our reward. To try and avoid some of these problems cooks often follow recipes. These are prescribed strategies, which we call policies in the more formal parlance of sequential decision problems, detailing the sequence of actions needed to produce a desired dish. Importantly, recipes often prescribe actions conditioned on the current state of the recipe which includes all information needed to decide on the next course of action. When making bread, for instance, the wet ingredients are typically introduced into premixed dry ingredients. The contents of the bowl, the dry ingredients and other prepared components of the recipe, are the current state of the world as it pertains to the cooking task. Whether you are familiar with cooking practice or not, I hope to convince you that sequential decision problems are part of our everyday experience (perhaps it is easy to formulate other examples with this one in mind). Moreover, these problems have special characteristics: the impact of decisions may not become clear until after many steps due to the property of *delayed consequences*, the world has a *state*, and agents interact with the world by executing actions prescribed by a policy.

In Artificial Intelligence and Machine Learning the subfield that focuses on sequential decision problems is called Reinforcement Learning (RL) (Sutton & Barto, 1998). In RL research the characteristics of sequential decision problems are formalized in Markov Decision Processes (MDPs). MDPs formalize a discrete time interaction between an agent and the domain it is interacting with. MDPs introduce a formal notion of state, action, delayed reward (consequences), and policies. In addition, MDPs introduce the concept of stochasticity and they define how a domain transitions between world states. Stochastic worlds exhibit random or imperfectly predictable behavior. Baked bread always turns out differently in the end despite our careful following of the recipe. It is arguable that this occurs because of the complex state (since we do not account for the

precise mixture of our ingredients), but we still need a tool accounting for the differences. The concept of stochasticity allows the MDP formalism to account for different responses to selected actions. The MDP framework provides a rigorously mathematical setting in which to study agents acting in sequential decision problems.

A basic problem in RL is to learn an optimal policy. Optimal policies maximize an explicit objective function (formally defined in Chapter 2). Unfortunately, agents in RL have limited knowledge of the domain they are interacting with, and therefore, the objective function is imperfectly known. To learn the optimal policy, the agent must execute experiments, try out potentially suboptimal actions, in search of the optimal policy. Good recipes are identified after extensive periods of experimentation. During experimentation, different combinations of ingredients are tried, and different processes of mixing and preparation are tried. All of these experiments are performed when we are not sure how best to proceed. Moreover, one is faced with deciding to try something new, perhaps discovering a better baking process, or taking advantage of what is already known by using a familiar recipe. In RL this problem is called the exploration/exploitation dilemma. It is a fundamental tradeoff in the study of RL agents.

In the case of human exploration, the selection of experiments is always influenced by prior knowledge. This knowledge can be gained in a variety of ways. For instance, we might have observed another cook prepare a similar meal, cooked something similar in the past, or have queried an expert about how to proceed in specific circumstances. Characteristically, humans exploit this knowledge to focus their experimentation.

This thesis focuses on exploiting the agent's past experience and interactions with an expert to address the exploration/exploitation dilemma in RL. To proceed, we need a formal method of expressing prior knowledge, a means of updating the knowledge when new information is made available, and a means of exploiting this knowledge in new

circumstances. The representation of knowledge must account for the variety of observations made by agents. Agents can explore by executing actions, querying an expert, observing expert demonstrations, and so on. Therefore, our formal representations of knowledge must support updating knowledge given observations of this kind. In addition, the representation must relate these past experiences to the formal properties of MDPs introduced in the following chapters.

The approach to knowledge representation taken in this thesis is explicitly Bayesian (MacKay, 2002; Jaynes, 2003). In the Bayesian framework, knowledge is formally defined by a prior distribution that specifies the agent's beliefs about the phenomena of interest. The basic question resolved by the Bayesian framework is how to update the agent's incomplete knowledge when new observations are available. Importantly, Bayesian theory provides an optimal means of updating the agent's knowledge such that decisions based on the updated knowledge are optimal. Bayesian methods have been used to study a variety of phenomena in human learning including perception, language, memory, and sensorimotor control (Shi & Griffiths, 2009; Körding & Wolpert, 2004; Ernst & Banks, 2002; Stocker & Simoncelli, 2007; Lee & Mumford, 2003). Evidence indicates that these unconscious processes conform remarkably to Bayesian explanations. Our focus is different. We focus on expressing specific forms of Bayesian prior knowledge to aid artificial agents in overcoming the exploration/exploitationg dilemma in MDPs. Thus, our contributions sit squarely within the growing literature on Bayesian RL, which we discuss in more detail in Chapter 2. Bayesian methods of knowledge specification are appealing in the RL setting because they lead to a theoretically justifiable solution to the exploration/exploitation problem. Below we discuss the specific contributions to the field of Bayesian RL made by this thesis.

1.1 Thesis Outline and Contributions

The remainder of this thesis has the following structure.

- In Chapter 2 we introduce the formal framework used to study sequential decision processing. Therein we formalize MDPs, the Multi-Task Reinforcement Learning (MTRL) setting (where agents interact with a sequence of MDPs), and Bayesian RL. As part of this chapter, we discuss the formal objectives of artificial agents interacting with MDPs, the nature of transfer in the MTRL setting, the specification of Bayesian prior knowledge for RL, and basic algorithms for each of these settings.

- In Chapter 3 we tackle the MTRL problem in which an agent is confronted with a sequence of related tasks. We contribute a hierarchical Bayesian model relating the agent's past task experiences with the current task. The model assumes that the set of tasks can be divided into related classes such that tasks belonging to the same class share similarities in their transition models. We define a method of posterior inference that learns the complete tasks structure including the number and kind of classes explaining the set of observed tasks. Additionally, we introduce an algorithm that exploits this knowledge to more effectively explore in new tasks. The algorithm uses the learned hierarchical structure as an informed prior distribution for Bayesian RL in the new task. We demonstrate that the transferred domain model knowledge improves the speed of learning in related tasks.

- Chapter 3 transferred knowledge of the transition function. In many cases it is more efficient to directly transfer solutions. In Chapter 4 we contribute a new algorithm, BPS, for direct policy search, which exploits a Bayesian prior distribution on the

policy space. Additionally, we contribute a hierarchical Bayesian model that captures policy components transferrable between related tasks. We develop a specific implementation of the BPS algorithm that learns the hierarchical policy model and exploits the policy knowledge in new tasks. We show that the hierarchical Bayesian model supports inferences from expert-generated data as well as agent-generated data. When provided with evidence from expert play, the agent can learn transferable policy components for reuse in a variety of tasks. If expert examples are not available, the agent can learn these transferrable components autonomously. We evaluate the algorithm in a complex multi-agent real-time-strategy game called Wargus.

- In the previous chapters we assumed a hierarchical relationship relating past experiences to new experiences. An alternative method of transferring knowledge is to explicitly represent similarity between solutions. Chapter 5 begins by introducing Bayesian Optimization. Bayesian Optimization is a general framework for global optimization of objective functions. It exploits a Bayesian model of the latent objective function to efficiently explore the objective function surface. The efficiency of this exploration depends on the quality of the objective function model that relates the values of policies in the policy space. We discuss the adaptation of the BO framework to the RL setting, and propose two distinct improvements to standard models used in BO for RL. Our new models exploit the sequential experience of Reinforcement Learning Agents to define improved measures of policy similarity. More accurate measures of policy similarity improve the generalization quality of the objective function model and reduce the number of experiments needed to identify the optimal policy. Empirically we demonstrate, in several benchmark

domains, that the new models do, in fact, reduce the number of samples needed to identify a near optimal policy.

- In Chapter 4 we showed that the BPS algorithm can learn optimal policies by observing expert demonstrations. Often one does not have access to an expert with control knowledge of this kind. Nevertheless an expert may still have sufficient knowledge to distinguish poor policies from good policies for a given task. The expert may be able to do this even when they cannot solve the task themselves. We propose exploiting expert knowledge of this kind. Chapter 6 considers the problem of learning control policies via trajectory preference queries to an expert. In particular, the agent presents an expert with short demonstrations originating from the same state and the expert indicates which trajectory is preferred. The agent's goal is to elicit a latent target policy from the expert with as few queries as possible. To tackle this problem, we propose a novel Bayesian model of the querying process and introduce two methods of exploiting this model to actively select expert queries. Experimental results on four benchmark problems indicate that our model can learn policies from trajectory preference queries and that active query selection can be substantially more efficient than random selection.

- Chapter 7 summarizes the contributions of the thesis.

Chapter 2: Background

2.1 Reinforcement Learning

To formalize the interaction between an agent and its environment, we turn to the mathematical theory of decision processes. In particular, we focus on decision processes formalizing discrete time interaction between an agent and the environment. The basic idea is shown in Figure 2.2. The agent observes properties of the world, receives rewards, and makes action selections as a function of its observations. Determining the best course of action to execute over many time steps is the central problem facing the agent. Given only the evidence of its past interactions, the agent aims to determine the most rewarding strategy to solve the current problem.

2.1.1 Markov Decision Processes

The work described in this thesis focuses on solutions to Markov Decision Processes (MDP). An MDP M is described by a tuple (S, A, P, P_0, R), where S represents a set of states of the world, A represents a set of actions available to the agent, the transition function $P(s_t|s_{t-1}, a_{t-1})$ defines the probability of transition to state s_t when action a_{t-1} is executed in state s_{t-1}, the initial state distribution $P_0(s_0)$ defines the probability of starting in state s_0, and the reward function $R(s_t, a_t)$ maps state-action pairs to a real-valued reward $r \in (-\infty, \infty)$. In this thesis, we consider MDPs with both discrete and continuous state and action spaces. The defining assumption of MDPs is the Markov

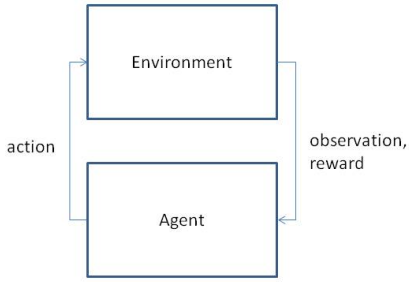


Figure 2.1: The discrete time interaction between an agent and its environment.

property which states the probability of transition to state s_t is independent of the history, conditioned on the current state and action. Concretely,

$$P(s_t|s_{t-1}, a_{t-1}) = P(s_t|s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, ..., s_0, a_0). \tag{2.1}$$

This statement has an important consequence. An agent acting with an MDP need only know the current state to make optimal action selections. There is no need to consider the agent's history.

An agent in an MDP executes a policy that defines how an agent selects actions in each state. A policy π is a mapping from the current state to an action, $\pi : s \mapsto a$. A stochastic policy π defines a distribution over actions $P_\pi(a|s)$ conditioned on the current state. A stochastic parametric policy is indexed by a vector of parameters θ. We write $P_\pi(a|s, \theta)$ to indicate the probability of selecting an action dependent on the state the

vector of policy parameters θ. Alternatively, we will write π_θ to indicate a parametric policy.

The challenge for RL agents is to assess the long term performance of their action selection policies. Agent's myopically focused on the immediate reward $R(s, a)$ may overlook policies with high payoff over longer time scales. Therefore, the agent must consider the long-term returns received when executing a policy. Let $r_0, r_1, r_3, ...$ be the sequence of rewards received starting from time step 0. The simplest method of measuring long-term reward is to consider the sum of rewards

$$r_0 + r_1 + ... + r_i + ...$$

observed by the agent. This quanitity is called the return. To be certain the return remains finite, additional assumptions must be introduced. In episodic domains, every sequence of rewards terminates in finite time with probability 1, and the return is defined to be,

$$D = r_0 + r_1 + ... + r_T,$$

where r_t denotes the immediate reward at time t, and T is the sequence length. Domains of this kind have natural points of termination. Examples include chess and soccer. Each of these examples has a set of starting conditions and rules terminating play after a finite period of time. Many other domains do not have this episodic characteristic. For these continuous tasks the return is discounted in time,

$$D_\gamma = r_0 + \gamma r_1 + \gamma^2 r_2 + ... = \sum_{i=0}^{\infty} \gamma^i r_i$$

For values of $\gamma \in [0, 1)$, discounting guarantees the return has a finite value.

To estimate the long-term performance of a policy, the agent must account for the stochastic transitions of the MDP. The inherent stochastic transition function, the initial state distribution, and the policy determine the observed sequence of states and the actions selected by the agent. Therefore, the return is a random variable, and its expected value measures the average quality of the agent's action selection policy. The expected return is defined to be

$$\eta(\pi) = E[r_0 + \gamma r_1 + ...|\pi] = E[\sum_{i=0}^{\infty} \gamma^i r_i|\pi].$$

This is the expected return the agent observes when starting in state s_0 and executing policy π. Alternatively, we write

$$V_\pi(s) = E[r_0 + \gamma r_1 + ...|\pi, s] = E[\sum_{i=0}^{\infty} \gamma^i r_i|\pi, s_0 = s]$$

to indicate the value of executing policy π starting from any specified state s. This quantity is called the **value function**. Closely related to the value function is the **Q-function**,

$$Q^\pi(s, a) = E[r_0 + \gamma r_1 + ...|s_0 = s, a_0 = a, \pi],$$

which is the value of executing policy π starting from state action pair (s, a).

In the undiscounted case, the expected return

$$\eta(\pi) = E[r_0 + \gamma r_1 + ...\gamma^{T-1} r_T|\pi] = E[\sum_{i=0}^{T} \gamma^i r_i|\pi]$$

we can re-express the undiscounted return by introducing trajectories. A trajectory $\xi = (s_0, a_0, ..., a_{T-1}, s_T)$ is a sequence of states and actions visited by the agent when

executing its policy. It follows from the definition of the transition probability function and the initial state distribution that the probability of a trajectory ξ given π is,

$$P(\xi|\pi) = P_0(s_0)\prod_{t=1}^{T} P(s_t|s_{t-1}, a_{t-1})P_\pi(a_{t-1}|s_{t-1}).$$

This is the probability of observing a trajectory ξ given that the agent starts in state s_0 and executes a policy π. The value of a trajectory,

$$R(\xi) = \sum_{t=0}^{T} r_t,$$

is simply the return of trajectory ξ. Consequently, the expected return can be expressed

$$\eta(\theta) = \int R(\xi)P(\xi|\pi)d\xi.$$

An important correspondence relates optimal expected returns and optimal policies. We define the optimal expected return to be

$$V^* = \max_\pi V_\pi(s),$$

and the optimal Q-function to be

$$Q^*(s, a) = \max_\pi Q^\pi(s, a).$$

These optimality equations entail the Bellman Equations

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^{\pi}(s'), \tag{2.2}$$

$$V^{*}(s) = \max_{a}(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^{*}(s')), \tag{2.3}$$

which give a recursive definition of the expected return. The second equation states that the optimal value function is computed by taking the best action and then acting optimally thereafter. Similarly, the expected return for a specific policy decomposes into the immediate reward for acting according to π plus the value of acting according to π thereafter.

The objective of a RL agent is to identify an optimal policy π^* that maximizes the expected return. In the MDP setting this policy always exists and can be computed as,

$$\pi^{*}(s) = \arg\max_{a}(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^{*}(s')).$$

However, computing the optimal policy using this formula is impractical when the state and action spaces are huge. For instance, in this thesis we will consider state spaces that decompose into a set of variables such that each state is a vector $s \in \Re^n$. Problematically, the number of states grows exponentially with the dimension n. This is called the curse of dimensionality, and it is a longstanding problem in RL. Consider a table-based representation of the value function that maintains a value for each possible state. Such a representation explicitly stores an exponential number of values. As the dimension of the state increases storing the value function in this way will quickly exceed available memory. Two solutions are possible. Introduce an approximate representation for the value function which generalize over states (Buşoniu *et al.*, 2011; Gordon, 1999). Alter-

natively, use parametric policy representations that do not require explicit representation of the value function. In this latter case, it is common to consider parameterized policy spaces $P_\pi(a|s,\theta)$. For a parametric policy to be optimal, it must satisfy the equation

$$\pi^*_\theta = \arg\max_\theta \eta(\theta), \tag{2.4}$$

which selects the parameters maximizing the expected return.

Finding the optimal policy is complicated by the agent's incomplete knowledge of the domain. In RL, the initial state distribution, transition function, and reward function are not provided to the agent. This gives rise to the **exploration/exploitation** problem. The agent must decide when it should take actions to learn something more and when it should exploit its current knowledge. Too much exploration will be costly to the agent, and too little will cause the agent to settle on a sub-optimal solution. A variety of strategies have been introduced to address this problem. Example strategies include epsilon-greedy which selects random actions with probability epsilon which is decreased over time (Sutton & Barto, 1998), optimistic policies that assume all untried actions lead to high reward (Brafman & Tennenholtz, 2003), and Bayesian approaches that remove the distinction between exploration and exploitation. In this thesis we will focus on Bayesian approaches which we discuss in more detail below.

2.1.2 Algorithms for MDPs

The body of work detailing algorithms for solving MDPs is expansive. In this sub-section we focus on a small set of standard algorithms and leave more specialized references to future chapters.

The Bellman equations state that an optimal value function defines an optimal policy. Therefore, a standard tactic in RL is to identify the optimal expected return. When the transition and reward function are known, a common algorithm called **value iteration** solves this problem (Sutton & Barto, 1998). To do so it exploits the recursive structure of the Bellman Equations by iteratively computing a sequence of updates

$$V_{k+1}(s) = \max_a (R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s')),$$

which converge to V^*. Value iteration terminates when $|V_{k+1} - V_k| < \epsilon$ for all states (where ϵ is close to zero). An alternative method called **policy iteration** iterates through a sequence of polices. Policy iteration first computes

$$V^{\pi_k}(s) = R(s, \pi_k(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi_k(s)) V^{\pi_k}(s')$$

for a given policy. Then it creates a new policy

$$\pi_{k+1}(s) = \arg\max_a (R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi_k}(s'))$$

that maximizes the computed value function. This process continues until the policy ceases to change. This occurs if and only if the policy is optimal.

These algorithms assume knowledge of the transition and reward function. Algorithms of this kind, which exploit estimates of the transition and reward function to compute the expected return, are called **model-based** algorithms. In some practical cases, it can be desirable to forgo explicitly learning the transition function. Algorithms which skip this step are called **model-free** algorithms. A standard example of a model-

free algorithm is Q-learning (Watkins & Dayan, 1992). The Q-learning algorithm learns directly from interaction with the environment. After each step taken by the agent, the Q-learning algorithm performs the following update

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(R(s,a) + \gamma \max_{a' \in A} Q(s',a'),$$

where α is a small step size parameter (that must be decreased over time). For discrete state and action spaces, and assuming proper update of the α parameter, the Q-learning algorithm is guaranteed to identify the optimal policy. By examining the Q-function update it is clear that this is done without estimating the transition function as an intermediate step.

All of these methods, both model-based and model-free, compute a value function to determine how to act in the domain. By contrast, direct policy search algorithms explore the space of policies directly. The most common forms of direct policy search are based on direct gradient ascent. The basic idea is to take repeated small steps

$$\theta_k = \theta_{k-1} + \nabla_{\theta_{k-1}} \eta(\theta_{k-1})$$

to iteratively improve the quality of the policy. The gradient can be estimated without knowledge of the transition function by using samples observed when executing π_{k-1}. Examples of algorithms using this approach include REINFORCE (Williams, 1992), and OLPOMDP (Baxter *et al.*, 2001). We will examine an alternative approach to policy search in Chapter 5 based on general techniques for global optimization.

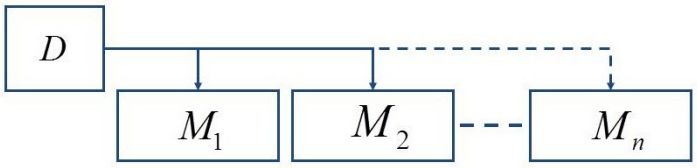


Figure 2.2: A sequence of MDPs generated by D.

2.2 Multi-Task Reinforcement Learning

The Multi-Task RL problem generalizes the standard RL setting. In the standard RL setting, the agent seeks an optimal policy for a single monolithic task. In the Multi-Task RL (MTRL) setting, agents solve a sequence of tasks.

To concretely define the MTRL setting, we propose the existence of a stochastic generative process D that outputs a sequence $M_1, \ldots M_n$ of MDPs. The subscript indicates the task number, and it is assumed that an agent acting in MDP M_i has experience in all MDPs $M_j : j < i$. Furthermore, we assume that the agent has explicit knowledge of task termination. Whenever the task changes, which moves the agent from task M_i to task M_{i+1}, the agent is made aware of this fact. This is important because it allows the agent to attribute observed changes in the MDP behavior to its current task.

In general, the generative process may output new MDPs with arbitrary structure. Such MDPs could differ in their state spaces, action spaces, transition function, and so on. In our work, the Generated MDPs may have distinct transition functions, reward functions, initial state distributions, but they share the same state and action spaces.

The evaluation of an agent in the MTRL setting is distinct from the standard RL setting. In the formal setting of MDPs, the goal of an RL agent is to maximize the expected return. This is only part of an MTRL agent's objective. In addition to this

goal, an MTRL agent seeks to benefit from its past experiences. If the MTRL agent has previous experience in i MDPs, the goal is to exploit the information extracted from these experiences to learn more quickly on MDP $i+1$ than an uninformed agent (which does not have prior experiences). In Chapter 3 and Chapter 4 we will discuss specific assumptions that lead to successful transfer in MTRL and we discuss the specifics of empirically evaluating our MTRL agents.

2.3 Bayesian Reinforcement Learning

We introduce the basic concepts behind Bayesian inference and discuss the use of Bayesian methods in RL.

Consider the basic problem of predicting a random variable Y using a conditional model $P(Y|Z)$, where Z is an unobserved variable. The goal will be to infer Z given measurements of Y. Assume that observations $O = (y_1, y_2, ...y_n)$ of Y are independently and identically distributed (iid). Under the iid assumption, the likelihood of the observations is $P(O|Z) = \prod_{j=1}^{n} P(y_j|Z)$. One way to infer the most likely value of Z is to maximize this likelihood function. Given the particular value of Z, maximizing the likelihood of Y proceeds straightforwardly.

The Bayesian approach assumes that the variable Z is unknown, but there exists knowledge of its distribution in the form of a *prior* $P(Z)$. Instead of a single explanation for Y, the Bayesian framework makes predictions $P(Y) = \int P(Y|Z)P(Z)dZ$ using all of this available knowledge. After observing O, the predictive distribution becomes $P(Y|O) = \int P(Y|Z)P(Z|O)dZ$, which is a function of the posterior distribution $P(Z|O)$. The posterior distribution encodes the probability Z given the observed data. It represents all available knowledge of the unobserved variable. In Bayesian statistics, the

process of learning reduces to computing this posterior distribution

$$P(Z|O) = \frac{P(O|Z)P(Z)}{P(O)} = \frac{P(O|Z)P(Z)}{\int P(O|Z)P(Z)dZ},$$

which combines prior, likelihood, and normalizing factors.

The key components of any Bayesian method are the specification of the prior distribution (encoding prior knowledge), and the means of posterior inference (either exact or approximate methods may be used). Computing the posterior distribution is complicated by the normalizing constant $P(O) = \int P(O|Z)P(Z)dZ$. In some special cases the prior distribution can be chosen such that the posterior has a simple closed form solution. For instance, using a *conjugate prior*, insures that the posterior and prior distributions are in the same family. However, in many cases the prior distribution does not have this convenient property, and computing the posterior distribution remains a serious problem.

In this thesis we introduce a variety of expressive prior distributions for use in Bayesian RL. Consequently, we do not have convenient posterior representations and must resort to approximations of the posterior distribution. For the purpose of approximating the posterior distribution, this thesis makes extensive use of Markov Chain Monte Carlo (MCMC) algorithms.

MCMC is a strategy for simulating a sequence of samples z belonging to a state space Z such that the sequence of generated samples tends to focus in "promising" regions of the state space. The mechanism for generating these samples is chosen such that the simulated sequence reflects a target distribution $P(Z)$. In other words, the sequence of samples generated using the MCMC strategy is, in the limit, indistinguishable from a set of samples generated directly from $P(Z)$. Obviously samples of this kind have a variety

of uses. For instance, it is straightforward to approximate the predictive distribution $P(Y|O) = \int P(Y|Z)P(Z|O)dZ \approx \sum_z P(Y|z)$ using the sample.

To generate these samples, MCMC algorithms produce a Markov chain $z_1, z_2, ...$ Transition between states is governed by a homogenous stochastic transition function $T(z^i|z^{i-1})$ defining the probability of moving to z_i from z_{i-1}. It is assumed that the transition function is a proper probability distribution such that for all states z_{i-1} $\int T(z^i|z^{i-1})dz^i = 1$. The homogenous property simply asserts that the transition function is invariant for all i. In addition, the transition function must be aperiodic and irreducible. The aperiodic property asserts that the chain cannot become trapped in cycles. The irreducible property asserts that for any two states z_i and z_j there is a path through Z such that $P(z_i|z_j) > 0$. MCMC sampling strategies are constructed such that the Markov chains they generate satisfy all of these properties and have $P(Z)$ as the target distribution. Below, we introduce the Metropolis Hastings (MH) sampler (Hastings, 1970; Metropolis *et al.*, 1953) and the Hybrid MCMC (HMC) sampler (Andrieu *et al.*, 2003a; Duane *et al.*, 1987), which are used extensively in this thesis.

We outline the MH sampler for target distribution $P(Z)$ in Algorithm 1. After initialization of state z, the MH algorithm proceeds in a sequence of steps. An MH step uses a proposal distribution $q(z_i|z_{i-1})$ to sample a candidate next state z_i given the current state z_{i-1}. The state of the chain moves in the direction of the newly sampled state with probability $A(x^*, x_{i-1}) = \min(1, \frac{P(z^*)q(z^*|z_{i-1})}{P(z_{i-1})q(z_{i-1}|z^*)})$. This simple procedure is the basis of many modern sampling algorithms which differ primarily in the construction of the proposal distribution. Careful selection of the proposal distribution is necessary to achieve high performance with Metropolis style algorithms. It is worth noting that the normalizing constant of the target distribution is not needed, because the ratio $\frac{P(z_i)}{P(z_{i-})}$ simplifies such that the normalizing constants vanish. This is an appealing property for

Algorithm 1 The Metropolis Hastings MCMC Sampler

Initialize z_0
for $i = 1 : N$ **do**
  Sample $u \sim U_{0,1}$
  Sample $z^* \sim q(z_i|z_{i-1})$
  if $u < A(x^*, x_{i-1}) = \min(1, \frac{P(z^*)q(z_{i-1}|z^*)}{P(z_{i-1})q(z^*|z_{i-1})})$ **then**
    $z_i = z^*$
  else
    $z_i = z_{i-1}$
  end if
end for

Bayesian posterior inference where the Metropolis sampler can be used to simulate the posterior distribution $P(Z|O)$ without needing to compute $P(O)$.

The HMC sampler aims to exploit gradient information to improve the quality of samples. After initialization, the HMC algorithm takes a step using the gradient of the energy function $log(P(Z))$. The HMC algorithm assumes that the state z is an n-dimensional vector $z \in \Re^n$. To generate samples, HMC introduces an auxiliary set of so called "momentum" variables d and an extended target density $P(Z, D) = P(Z)N(D; 0, I^n)$ where $N(D; 0, I^n)$ is a multivariate Gaussian distribution with diagonal covariance I^n. To generate a state of the Markov chain HMC generates the Gaussian random variable d^* and then takes a step in z and d. The updates to z and d are governed by the scaling parameter ρ and incorporate the gradient information as shown in Algorithm 2. The updated state variables are accepted with probability $A = \min(1, \frac{P(z_i)}{P(z_{i-1}} \exp(-\frac{1}{2}(d_1^T d_1 - d^{*T} d^*)))$. This procedure is a special case of the MH algorithm, and the sequence of states z can be shown to reflect the target distribution $P(Z)$. Like the MH algorithm above, it is not necessary to know the normalizing constant of the target distribution to use the HMC sampler. HMC is particularly appropriate when the dimensionality of the state space is high. In this case, undirected sampling procedures can exhibit poor mixing capabilities

Algorithm 2 The Hybrid MCMC Sampler

Initialize z_0
for $i = 1 : N$ **do**
  $u \sim U_{0,1}$ and $d^* \sim N(0, I^n)$
  $z_0 = z_{i-1} d_0 = d^* + \frac{\rho}{2} \nabla \log(P(z_{i-1}))$
  $z_0 = z_{i-1} + d_0$
  $d_1 = d_0 + \rho \cdot \nabla \log(P(z_i))$
  if $u < A = \min(1, \frac{P(z_i)}{P(z_{i-1}} exp(-\frac{1}{2}(d_1^T d_1 - d^{*T} d^*)))$ **then**
    $(z_i, u_i) = (z_0, d_1)$
  else
    $(z_i, u_i) = (z_{i-1}, d^*)$
  end if
end for

(requiring many samples before the target density is represented). The gradient information helps to reduce the random walk behavior of the Gaussian proposal density and thereby improves the mixing property of the algorithm in high dimensions.

Additional details of our sampling algorithms, which modify these basic procedures, can be found in the following chapters.

Bayesian Methods in RL. We will consider two basic approaches to exploiting Bayesian prior distributions in the RL setting: model-based and model-free. In the Bayesian model-based setting, prior distributions are defined over the parameters of the transition function (and possibly the reward function) (Dearden *et al.*, 1998; Strens, 2000; Asmuth *et al.*, 2009; Wang *et al.*, 2005; Duff, 2003; Rasmussen & Kuss, 2004; Poupart *et al.*, 2006b; Deisenroth & Rasmussen, 2011). In the model-free setting, the prior distribution is over the value function (Dearden *et al.*, 1999; Ghavamzadeh & Engel, 2007a; Engel *et al.*, 2003; Engel *et al.*, 2005; Lizotte *et al.*, 2007), the value function gradient (Ghavamzadeh & Engel, 2006; Ghavamzadeh & Engel, 2007b), or directly over the policy (Hoffman *et al.*, 2007).

Model-Based Bayesian RL. In model-based Bayesian RL, the agent maintains a belief about which MDP it is in. The transition function, reward function, and starting state distributions are governed by a set of parameters ϕ. A particular setting of the transition function parameters defines an MDP by specifying the probability of transition between states. The agent's belief is represented by a prior distribution $b(\phi)$ encoding the a-priori probability that the agent is in a particular environment. As an agent explores its environment, it witnesses transition tuples $O = \{(s, a, s')\}$ and updates its beliefs about ϕ via Bayesian posterior inference.

Consider an agent that finds itself in state s and executes action a. After transition to the next state s, two things have changed. The agent has observed itself in a new state, and the agent's beliefs have changed to account for the newly observed transition and reward. Therefore, transition in the Bayesian model-based setting occurs between joint information-states that are composed of real states s and belief states b. The probability of transition to s' from s is $P(s'|s, a, b) = \int b(\phi)P(s'|s, a, \phi)d\phi$, and the transition between belief states is written

$$P(b'|s, a, s', b) = \begin{cases} 1 & \text{if } b' = b(\phi|s, a, s') \\ 0 & \text{otherwise} \end{cases}$$

(b' is the Bayesian posterior distribution). Taken together these equations define the transition function $P(s', b'|s, a, b) = P(b'|s, a, s', b)P(s'|s, a, b)$ for a belief state MDP.

In Bayesian model-based RL, there are corresponding definitions for the optimal value function of belief state MDPs,

$$V^*(s, b) = \max_a (R(s, a) + \lambda \sum_{s' \in S} P(s'|s, a, b)V^*(s', b')), \tag{2.5}$$

and the optimal Q-function,

$$Q^*(s,a,b) = R(s,\pi(s)) + \lambda \sum_{s' \in S} P(s'|s,a,b)V^*(s',b'). \tag{2.6}$$

These definitions incorporate the agent's changing beliefs into the recursive Bellman optimality equations. In the previous section introducing MDPs, it was noted that the solution of the Bellman equations yields an optimal policy $\pi^*(s) = \arg\max_a(R(s,a) + \lambda \sum_{s' \in S} P(s'|s,a)V^*(s'))$. Similarly, the solution of the belief state Bellman equations yields an optimal policy as well,

$$\pi^*(s,b) = \arg\max_a(R(s,a) + \lambda \sum_{s' \in S} P(s'|s,a,b)V^*(s',b')).$$

Intuitively the agent selects actions based on what it currently knows, according to its belief state, and based on the impact its actions will have on its future knowledge. The equation recursively considers the impact of observations on the agent's beliefs. This formulation has an important consequence for the exploration/exploitation dilemma in RL. It is no longer necessary to introduce a separate exploration strategy. The Bayes optimal policy optimally explores the MDP. This is a compelling theoretical property, because it resolves this longstanding issue in RL. Unfortunately, exactly solving Equation 2.5 is computationally intractable due to the number of belief states (Duff, 2003; Poupart *et al.*, 2006b).

Work on model-based Bayesian RL can be decomposed based on the specified prior distributions and the means of approximating the Bayes optimal policy. Much of the literature has focused on simple belief models. For instance, in MDPs with discrete state spaces, simple Dirichlet priors are commonly used (Dearden *et al.*, 1998; Strens, 2000;

Asmuth *et al.*, 2009; Wang *et al.*, 2005; Duff, 2003; Poupart *et al.*, 2006b). However, some notable exceptions which introduce novel priors for structured and continuous MDPs. For instance, Ross and Pineau (2008,2012) introduced prior distributions for MDPs with factored state spaces. [1] Work by (Rasmussen & Kuss, 2004; Deisenroth & Rasmussen, 2011) employed Gaussian Process (Williams & Rasmussen, 1996) models of the transition function in continuous state space MDPs. Work by (Ross *et al.*, 2008) used multi-variate Gaussian Wishart distributions for a robot navigation task. Whichever model is used, the problem of estimating Equation 2.5 remains.

A variety of heuristic action selection policies have been introduced in the Bayesian RL literature. For instance, the work by (Dearden *et al.*, 1999) considers the expected gain for selecting a particular action. They introduce a heuristic method of action selection based on the Value of Perfect Information. Work by (Strens, 2000) uses a Thompson Sampling heuristic. This heuristic samples a single model from the posterior distribution $b(\phi)$, solves the Bellman equation for this model, and then greedily selects an action using the corresponding optimal policy. Similarly, (Asmuth *et al.*, 2009) samples a collection of models from the posterior distribution, combines the sampled models into a single model, computes the value function for the combined model, and then selects the greedy action. Alternatively, some algorithms explicitly attempt online approximations. Work by (Wang *et al.*, 2005) adapted the Sparse Sampling (Kearns *et al.*, 2002) algorithm to the Bayesian setting. They sample a sparse set of paths and use this sampled set to estimate Equation 2.5. Similarly, (Ross & Pineau, 2008) used Monte-Carlo roll out planning to approximate the belief state value function. An alternative approach introduced by (Poupart *et al.*, 2006a) suggested an explicit piecewise polynomial approximation of

[1] The states of factored MDPs decompose into a set of variables (Guestrin *et al.*, 2003). It is typical to represent the transition models using Dynamic Bayesian Networks (Heckerman *et al.*, 1995).

Equation 2.5. Alternatively, some strategies use the variance of the model parameters as shaping rewards (Sorg *et al.*, 2010) that direct the agent to states where it is least certain of the true model.

Model-Free Bayesian RL. As discussed in the previous chapter, model-free methods attempt to estimate the value function without explicit representation of the transition function. Intuitively, the model-based approach discussed above indirectly expresses a distribution over the value function. Consider a single sample from the belief $b(\phi)$. This sample specifies a particular transition function and a corresponding optimal value function. Hence, the distribution $b(\phi)$ can be thought of as indirectly specifying a distribution over optimal value functions. The goal in model-free Bayesian RL is the representation of value function uncertainty without explicit representation of transition function uncertainty.

Uncertainty in the value function arises due to the limits of experience imposed on RL agents. Typically the agent cannot observe the value function directly. Instead it observes the return D, which is random due to the stochastic transitions of the MDP and the agent's policy. Given these restrictions, the value function is imprecisely known. Bayesian methods quantify this uncertainty by constructing priors $P(D)$ such that the expected value of the return $E_{P(D)}[D]$ is proportional to the value function or Q-function.

A variety of prior models for model-free Bayesian RL have been introduced in the literature. For instance, (Engel *et al.*, 2005; Engel *et al.*, 2003) tackles the problem of Bayesian Q-function estimation. They specify a Gaussian Process prior distribution $P(D_\pi)$, where D_π is the return for policy π, such that $E_{P(D_\pi)}[D_\pi] = Q(s, a)$. To optimize the policy, they propose an algorithm based on SARSA (Sutton & Barto, 1998) that exploits the Q-function model to perform policy improvement steps (similar to the policy iteration algorithm introduced earlier). The work presented in (Dearden *et al.*,

1998) presents a Bayesian model of the optimal Q-function for discrete state and action spaces. By contrast to the previous work this model directly estimates the optimal Q-function. To do this, a Gaussian distribution is associated with each state-action pair, and the distribution is encoded such that $E_{P(D)}[D] = Q^*(s, a)$. A heuristic method of action selection that measures the gain in information for executing an action is used to approximate the Bayes optimal action selection policy.

Alternatively, it is possible to optimize parametric policies in the Bayesian model-free framework. For instance, (Ghavamzadeh & Engel, 2006; Ghavamzadeh & Engel, 2007b) decomposes the gradient of the expected return

$$\nabla\eta(\theta) = \int R(\xi)\nabla log(P(\xi|\theta))P(\xi|\theta)d\xi.$$

and proposes modeling these components. They consider Gaussian process models for $R(\xi)\nabla log(P(\xi|\theta))$ and $R(\xi)$. Given models of these functions, they derive distributions for $\nabla\eta(\theta)$. The computed gradient models are applied to update the parametric policy parameters in the direction of the posterior mean of the gradient of the expected return. A related approach models the expected return $\eta(\theta)$ directly. For instance, (Lizotte *et al.*, 2007; Lizotte, 2008b) proposed a Gaussian Process model of $\eta(\theta)$. By contrast to previous work, it is assumed that a Monte-Carlo estimate of the expected return can be directly observed. Using the posterior model, they construct a heuristic function $H(\theta)$ that evaluates the quality of policy π_θ. The policy maximizing $H(\theta)$ is executed, the new data is observed, and a new policy is selected. This procedure repeats until the optimal policy is identified.

The discussion above is not an exhaustive review of the field of Bayesian RL. Instead it is intended to organize the work in Bayesian RL based on the kind of prior knowledge,

the prior models, and the method of action selection. Our contributions will include both model-based and model-free algorithms, we will specify new prior distributions for both the model-based and model-free settings, and we will contribute new action selection procedures. In the following chapters, we will provide additional explanation of related work to more concretely highlight the contributions therein.

Chapter 3: Hierarchical Bayesian Transfer for Model-Based Reinforcement Learning

Complete knowledge of the MDP model reduces the reinforcement learning problem to planning. Given the model, the optimal policy can be deduced from internal deliberation facilitated by model predictions. Availability of the model implies considerable knowledge of the domain such that the agent can answer arbitrary questions regarding the possible repercussions of action selections. In this chapter, we discuss the transfer of domain model knowledge. All other components of the MDP, the action space, state space, and policy space, are shared. The primary contributions of this chapter include a specific hierarchical model relating the domain model knowledge from distinct tasks, a Bayesian prior expressing a priori uncertainty of the hierarchical model structure, and an algorithm which uses the Bayesian model to advantage when confronted with related tasks.

As a motivating example, consider a foraging agent who is interested in finding food. Depending on the type of the environment, the indicators for low cost paths to food sources might differ. Perhaps, in some environments, it is safest to travel from rock to rock. In other environments, food may be more abundant along river beds, and a different strategy for selecting paths will be needed. Moreover, the agent does not know a priori what the indicators for low cost paths might be and what type of environment it currently is in.

We would like the agent to learn the distribution of different types of environments

and the indicators of low cost paths in each environment after exploring a few such environments. After this, given a small amount of experience in a new environment, the agent should be able to quickly make inferences about the environment type, and use that information to explore more efficiently. For example, if an agent finds that there is no food near the rocks, it might conclude that the river beds might be better places to explore next, and choose its path accordingly. Benefitting from transfer is of special importance for foraging animals. Choices of foraging paths influence the chances of predation. Quickly identifying the lowest cost foraging paths is necessary.

A naïve agent may confuse its path preferences by treating all of its observations as coming from a single MDP. In this case, the agent would estimate the most likely set of MDP parameters that generate the given observations. If in fact the MDPs generating the observations are not similar to each other, this might hurt exploration in a new MDP, because the learned model will generalize poorly. It may even result in slower learning than if the agent assumes nothing about the model. Our goal is to take advantage of the similar structure between MDPs when it exists, while at the same time avoiding or limiting knowledge transfer between dissimilar MDPs. We do this by introducing a hierarchical infinite mixture that introduces mixture components to explain the available data. Each component captures uncertainty in both the MDP structure and parameter values and stands for a class of "similar MDPs." After exploring a set of MDPs, the agent learns the distribution of different classes and their model characteristics. We evaluate our approach on two domains inspired by rodent foraging experiments and demonstrate that (i) our approach can use the learned hierarchical model to explore more efficiently in a new environment than an agent with no prior knowledge, (ii) it can successfully learn the number of underlying MDP classes, and (iii) it can quickly adapt to the case where the new MDP does not belong to a class it has seen before.

3.1 Related Work

Knowledge transfer in the MTRL setting has attracted a great deal of interest, and several methods have been proposed that enable RL agents to take advantage of solutions and observations from prior tasks. Some of these methods assume that successive tasks share similar dynamics and reward structure, for example methods that use approximations of prior optimal value functions as initial value functions for new tasks (Konidaris & Barto, 2006). Other approaches attempt to automatically construct general features that enable mapping substantially different tasks together, and transferring value functions defined in terms of these features (Banerjee & Stone, 2007). Yet other approaches (Mehta *et al.*, 2005) assume that the reward function of the new task is given, maintain value functions for several different prior tasks, and initialize the value function of a new task with one that yields the most reward. The current work differs from these previous approaches in that we adopt a more principled hierarchical Bayesian framework to formalize the similarities between the different MDPs. By maintaining probabilistic models at multiple levels of the MDP class hierarchy, we can naturally take advantage of the common structure between different RL tasks, while also learning specialized knowledge in a single task.

Hierarchical Bayesian RL is also related to Bayesian Reinforcement Learning (Dearden *et al.*, 1999; Dearden *et al.*, 1998; Strens, 2000; Duff, 2003), where the goal is to give a principled solution to the problem of exploration by explicitly modeling the uncertainty in the rewards, state-transition models, and value functions. Here, a distribution is maintained over possible true MDPs rather than a single maximum likelihood estimate. This distribution is used to select actions of high expected utility. However, in standard Bayesian RL, it is difficult to choose an informed prior distribution over

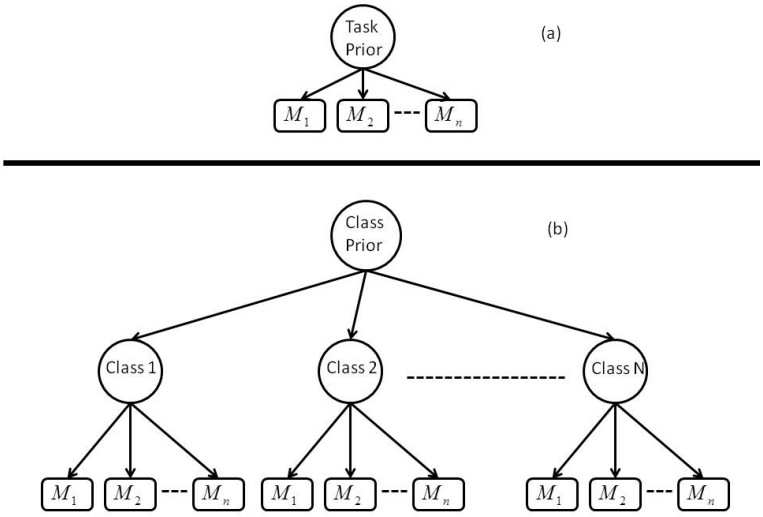


Figure 3.1: (a) Single task Bayesian RL vs. (b) Hierarchical Bayesian MTRL. The number of classes N and the parameters of the class distributions are learned from data.

MDPs. In the MTRL setting, however, a Bayesian approach facilitates knowledge transfer across MDPs by providing a much more informed starting point. Thus, we believe the multi-task setting and our hierarchical approach better highlights the potential utility of Bayesian RL.

3.2 Overview

Our approach to multi-task reinforcement learning can be viewed as extending Bayesian approaches to RL to a multi-task setting. In (single-task) Bayesian model-based RL, a posterior probability distribution $P(M|\Theta, O)$ is maintained over possible MDPs, where M denotes a random variable over MDPs, O is the current set of observations from the underlying MDP, and Θ denotes the parameters of the probability model (Figure 3.1(a)).

This distribution is used to select actions and is refined as the agent acts and observes the environment.

Our MTRL approach follows this Bayesian framework and adds to it the key idea of using hierarchical Bayes to model "classes" of MDPs. Figure 3.1(b) depicts the general form of our generative model. We introduce a (hidden) random variable C over "possible MDP classes," where each class $C = c$ in turn defines a distribution over the probability parameters Θ above. Sampling from this induced distribution results in an MDP $M = m$ with probability $\Pr(M = m|\Theta = \theta, C = c)$. This addition of a "class layer" makes the probability model hierarchical. Intuitively, the hierarchical Bayesian approach allows us to explicitly reason about the shared structure of certain MDPs and to transfer knowledge to a new MDP from those in the same class.

Algorithm 3 Hierarchical Bayesian MTRL Algorithm

1: Initialize the hierarchical model parameters Ψ
2: **for** each MDP M_i from $i = 1, 2, \ldots$ **do**
3: $O_i = \emptyset$ //O_i is the set of observations for M_i
4: **while** policy π has not converged **do**
5: $\hat{M}_i \leftarrow$ `SampleMAP`$(\Pr(M \mid O_i, \Psi))$ // Section 4.4
6: $\pi =$ `Solve`$(\hat{M}_i)$ //e.g. by value iteration
7: `Run` π in M_i for k steps
8: $O_i = O_i \cup \{$observations from k steps$\}$
9: **end while**
10: $\Psi \leftarrow$ `SampleMAP`$(\Psi|\hat{M}_1, \ldots, \hat{M}_{i-1})$ // Section 4.3
11: **end for**

To achieve the above behavior, it is critical that our system be able to automatically learn the class-level model and to allow for class inference of newly observed MDPs. Therefore, we utilize a nonparametric infinite mixture model at the class layer. This allows us to (a) model a potentially unbounded number of classes, (b) adapt to and make inferences about classes we have not seen before, and (c) learn the shared structure that

constitutes each class on the fly, rather than providing a rigid specification for each class (of course, the shared structure needs to conform to the assumptions we make about the probabilistic model).

The key steps in our approach are outlined in Algorithm 3. Our algorithm uses a hierarchical Bayesian model to estimate the MTRL distribution over MDPs, which is used as a strong prior for Bayesian RL in a new MDP. Initially before any MDPs are experienced, the hierarchical model parameters Ψ are initialized to uninformed values. The procedure SampleMap (line 5) generates a sample of MDPs according to $\Pr(M|\Psi)$ and picks one, $\hat{M}_i$, that has the highest probability in the sample. $\hat{M}_i$ is then solved (for example, using value iteration) for the optimal policy π (line 6) and then π is followed for k steps in the environment (line 7). This is similar to Thompson sampling, which has been employed with success in prior work (Thompson, 1933; Strens, 2000; Wang *et al.*, 2005). The observations gathered during those k steps are then stored in the observation database O_i for M_i. The agent then applies O_i to update the posterior distribution $\Pr(M|\Psi, O_i)$, and computes a new $\hat{M}_i$. This loop is repeated for MDP $\hat{M}_i$ until the policy has converged. After convergence, we update the hierarchical model parameters Ψ based on the parameter estimates $\hat{M}_i$. This is also done using the SampleMap procedure, where the sample is generated using the Gibbs sampling procedure described in Section 4.3. It uses the conditional distributions in Equation 2 to assign (possibly new) classes to each observed MDP $M_1, \ldots, M_i$. Given the assignments, we compute the parameters associated with each class distribution. This process is iterated until convergence. Intuitively, this step will result in a Ψ that captures the inherent class structure of the sequence of MDPs before and including M_i.

In the following, we give details on each step of our approach. First, we describe our hierarchical probability model. Next, we describe how we compute the model parameters

Ψ given a sequence of previously experienced MDPs, followed by the sampling procedure for an MDP given the model parameters Ψ and observations.

3.3 Hierarchical Bayesian Model

We propose a generative model corresponding to the hierarchical class structure illustrated in Figure 3.1(b). Our model decomposes the set of observed MDPs into classes. Each class $C = c$ is associated with a vector of parameters θ_c. These parameter vectors define distributions from which individual MDPs will be drawn. Drawing an MDP means specifying the parameters of its reward and transition model. Therefore, the class distributions represent prior uncertainty of the parameter values. When discussing our hierarchical model in the proceeding sections, we do not indicate specific forms for the class prior nor the MDP class distributions. For purposes of the current discussion, we need only assume it is possible to sample from the necessary distributions. Later, in the experimental results presented in this chapter, we will propose specific distributions appropriate for the test cases.

The generative process proceeds in a straightforward way. First, a class of MDPs is sampled from the class prior distribution. Subsequently a new MDP is sampled from the class distribution. Finally, observation data associated with the sampled MDP are generated as the agent executes actions. We model this generative process using a Dirichlet Process distribution. The DP prior has a useful interpretation in terms of an infinite mixture of components. An illustration of this interpretation is depicted using plate notation in Figure 3.2. Rectangles indicate probability distributions that are replicated a certain number of times, shown in the bottom-right corner. The distribution G_0 represents the class prior distribution. It is the agent's high level knowledge of the

class structure. The individual class distributions are associated with the parameters θ_c. There are a countably infinite number of potential classes. This feature of the DP allows the model to adapt the number of classes to the data by selecting a subset of classes from the countably infinite set which best describe the observations. Class distributions represent knowledge specific to a collection of MDPs. Associated with each MDP M_i is its class indicator c_i. The class indicator acts as an index into the set of available class distributions. Its setting indicates which class distribution is associated with a given MDP. In each MDP the agent is allowed R interactions resulting in tuples of observation data. Note that observation data is bagged with individual MDPs and is not shared across MDPs. This insures it is possible to learn MDP model parameters specific to each task. Finally, the parameter α is referred to as the concentration parameter. The DP implements a "rich get richer" scheme for allocating data to class components (highly populated components are more likely to be associated with new data points). The concentration parameter influences the probability of introducing new components to explain the growing data set. Higher values increase the likelihood of new components, and lower values have the opposite effect. Taken together, we denote the complete set of parameters $\Psi = (G_0, c, \theta, \alpha)$. Below we discuss the benefits of the DP model in more detail and concretely specify the method we employ for posterior inference.

To understand the role of the Dirichlet Process, first consider the case when there are a finite, known number of MDP classes. Suppose we observe a set of N MDPs from a set of k classes. In this case, it is natural to use a multinomial model over class assignments, with parameter vector β. The probability of the observed data will be proportional to $\prod_{j=1}^{k} \beta_j^{c_j}$ where c_j is the number of MDPs observed to be in class j and $\sum_j c_j = N$. If we wish to infer β, we can use an informative Dirichlet prior. The Dirichlet prior indicates our prior belief about the prevalence of each class in the data—if we expect that most

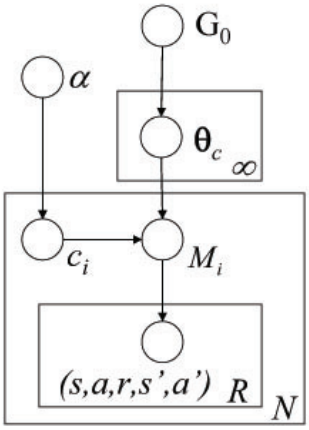


Figure 3.2: Infinite Mixture Model (Parameter set Ψ). There are N tasks. For each task the agent has made R observations. The class distributions are parameterized by θ_c (level 2 in Figure 3.1) and the distribution G_0 is a hyperprior over classes (the root in Figure 3.1). The class assignments c_i assign each MDP to a class. The parameter α influences the probability of introducing a new class to explain the existing data.

MDPs will be drawn from class c, the Dirichlet parameter (count) corresponding to c should be large. This will mean that our initial MAP estimates of β_c will be larger than for other classes.

Now consider the case when we do not know how many classes there are. In this case, we cannot explicitly estimate or use the multinomial distribution, since the number of parameters $|\beta|$ is unknown. However, using Dirichlet Process priors, it is possible to estimate the *conditional distribution* of the class of a new observation given the class assignment for previous observations, using the equations:

$$\begin{aligned} \Pr(C_i = c | c_1, \ldots, c_{i-1}) &\propto \frac{n_{-i,c}}{N - 1 + \alpha} \\ \Pr(C_i \neq c_j \forall j < i | c_1, ..., c_{i-1}) &\propto \frac{\alpha}{N - 1 + \alpha} \end{aligned} \tag{3.1}$$

Here, C_i a random variable over possible class assignments to the i^{th} data point, $c_1, \ldots, c_{i-1}$ are class assignments to the previous $i-1$ observations, $n_{-i,c}$ is the number of data points, excluding point i, currently assigned to class c. The parameter α governs the probability with which the Dirichlet Process hypothesizes that the i^{th} observation comes from a new class (sometimes called an "auxiliary class"). Using Equation 3.1, we can define a Gibbs sampling procedure that repeatedly samples class assignments until convergence. Observe that this procedure implicitly defines a multinomial distribution, but now the parameter vector for this distribution can vary in size as we observe more data[1]. Thus, the Dirichlet Process prior can be used to specify a distribution over an unbounded, potentially infinite set of classes from which our observations are drawn.

3.4 Updating the Hierarchical Model

In this section, we describe how we update the hierarchical model parameters Ψ (line 10 in algorithm 3) given a set of parameter estimates $\hat{M}_1, \ldots, \hat{M}_{i-1}$ and a new parameter estimate for the i^{th} MDP, $\hat{M}_i$. These estimates are treated like observed data d-separated from the observations O. It is difficult to compute a closed-form MAP estimate for Ψ given the observations due to the exponential number of joint class assignments. Therefore, we use a Gibbs sampling routine (Neal, 2000) described in Algorithm 9 to simulate samples from the posterior distribution $\Pr(\Psi|\hat{M}_1, \ldots, \hat{M}_{i-1})$, and we select the sample that maximizes the posterior probability.

Algorithm 9 is an auxiliary class Markov Chain Monte Carlo sampling technique for Dirichlet Process models where the hyper-prior distribution G_0 is not conjugate

[1]For more detailed discussion on the relationship between finite mixture models and Dirichlet Process priors please see (Neal, 2000).

Algorithm 4 Auxiliary Class Gibbs Sampler

Let m = Number of auxiliary classes.
Let F be the MDP distribution given a class: $M \sim F(\theta_c)$
Let M_j be the final estimates for the MDP j's parameters, for all j
Initialize the Markov chain state ($\Theta = \Theta_0, C = C_0$)
repeat
  Let $K = |\Theta|$
  for $c = K + 1 : K + m$ **do**
    Draw θ_c from G_0
  end for
  Let $\hat{\Psi} = \{C, \Theta, G_0, \alpha\}$
  for j=1:i **do**
    $c_j =$ `SampleAssignment`$(\hat{\Psi}, M_j, F, m)$
  end for
  Remove all classes with zero MDPs.
  $\Theta \leftarrow$ Sample($\Pr(\Theta'|c_1, \ldots, c_i)$)
until convergence

to the component distribution (we have adapted Algorithm 8 from (Neal, 2000)). In our case, the state of the Markov chain includes the set of class assignment variables $C_0 = \{c_1, c_2, \ldots, c_i\}$ and the class parameters $\Theta_0 = \{\theta_1, \theta_2, \ldots, \theta_k\}$ (if we have seen k classes so far). Simulation from the posterior distribution decomposes into two basic parts: simulation of the class assignment variables and simulation of the class parameters. Each of these sets is updated in turn. First, given the class parameter assignments of the MDPs, new class assignments are sampled. Second, given the assignments, new class parameters are sampled. Each iteration completely updates the state of the Markov chain.

The algorithm works as follows. The Markov chain is initialized by allocating all observations to a single class randomly generated from G_0. After initialization, the `SampleAssignment` routine (Algorithm 5) is used to propose a new class assignment for each of the observed MDP models. The probability of assigning M_i to class c in the

`SampleAssignment` function is proportional to Equation 3.1 and the posterior component likelihood $F(\theta_c, M_i)$ of the MDP model M_i. There are two possible cases for assignment. Either the MDP model is assigned to an existing class, in which case the parameters θ_c are already known, or the MDP is assigned to one of a fixed number of auxiliary classes (we found setting the number of auxiliary classes m to a small value resulted in good performance), in which case the auxiliary class parameters θ_c are randomly generated from the class prior G_0. The introduction of the auxiliary components allows the set of classes to grow automatically to fit the data. Effectively, the prior generates m alternative explanations for model M_i. The simulation procedure iterates through each of the MDPs and samples associated assignments. As a consequence of this iterated assignment procedure some classes may ultimately have zero MDPs assigned to them. These classes are deleted so that only classes with assigned data remain for the next step. The next step of the sampling procedure updates the class parameter vectors. Given the assignments, a new set of class parameters are sampled from $\Pr(\Theta'|c_1, \ldots, c_i)$, which decomposes into the product of component distributions $\prod_c P(\theta_c|M_i : c_i = c)$.

The class assignments and class parameters are sampled in sequence until we have a sufficiently large sample from the posterior. The assignment, $\Psi = (C, \Theta)$, that maximizes the posterior probability is then returned. This state represents a fixed decomposition of the existing MDPs into a set of classes (the decomposition is encoded in the assignment parameters) and the set of class parameter values explaining the observed MDPs (each $\theta_c \in \Theta$ encodes the necessary class information).

The ability of our algorithm to transfer is dependent on the true set of classes being small. Given a good decomposition of MDPs into classes the agent can determine, quickly, to which class the new environment belongs, and can exploit this information to improve its action selections. Alternatively, if the new environment does not belong with

any of the existing classes the agent must learn new models to explain its observations.

3.5 Transfer of Model Knowledge

In this section, we describe the procedure for sampling an MDP from the current hierarchical model for purposes of action selection in a new task. Actions are executed at each state of the MDP. Therefore, the sampled hypothesis model parameters must be sampled after each new observation. To do this, at each step, the agent must have a good idea about the identify of the class from which the current MDP is drawn. A computationally expensive method of maintaining the accuracy of the current hypothesis is to update the parameter estimate, using Algorithm 9 above, after each observation. However, this is unnecessary in practice. We need only sample the full posterior between environments and use the sampled structure, including the class assignments C, and class parameters Θ as an informed prior. Effectively, we decompose the state of the Markov chain into two parts. The first part of the state is the fixed hierarchy returned from Algorithm 9 explaining the previously observed MDPs. The second part of the state is the assignment, class parameters, and MDP model associated with the new task. It is only the latter portion of this state that is discussed in this section. To take advantage of the fixed portion of the Markov state the agent need only determine to which current class, if any, M_i belongs. If M_i belongs to one of the known classes, ($c_i \in 1..k$), then the information in θ_{c_i} is useful for exploration. Otherwise, the agent performs no worse than if it used the prior G_0.

To do this properly, we need to maintain a sample $\hat{M}_i$ for the MDP model parameters, and a current hypothesis for the class assignment. In the previous section, we directly decomposed the set of MDP model parameters. These values were estimated from the

associated observations and treated like observable data during posterior simulation. In the new MDP, we do not have this luxury. The available data are insufficient to fully specify the model parameters. Therefore, we must treat the model parameters as unobserved random variables.

Intuitively, simulation proceeds as per Algorithm 9 with the added step of updating the MDP model parameters. First, we initialize $\hat{M}_i$ by sampling from the informed prior. To do so we sample a class assignment at random, and then sample an MDP model from the class distribution. After initialization simulation of the state decomposes into three parts. First, the algorithm samples an assignment using Algorithm 5 for $c_i = c$. This portion of the algorithm assigns the current MDP to one of the existing classes or an auxiliary class sampled from the class prior. After sampling the class assignment Algorithm 9 would proceed to re-sampling the set of class parameter vectors. However, in this case, it is not necessary to re-sample class parameters, this portion of the state is fixed, unless the MDP has been assigned to an auxiliary class. In the case of an auxiliary class the MDP parameters are sampled from G_0. Once the MDP has been assigned to a class the algorithm samples the MDP model parameters from the posterior distribution $P(M_i|\theta_c, O_i)$. The conditional posterior distribution has this simple form because the class parameters θ_c have been fixed by the previous sampling steps. These three basic steps, sampling a class, sampling class parameters (if the class is auxiliary), and sampling the MDP model are repeated until the Markov chain converges to the posterior distribution.

After simulation, we use the sequence of sampled states to select a new MDP model and choose an action. Given the sequence of states, the algorithm selects the most probable class assignment c, and then samples an MDP from class c using the posterior distribution $P(M_i|\theta_c, O_i)$. To determine the most probable class assignment, we select

Algorithm 5 Sample Assignment $(\hat{\Psi}, M_j, F, m)$

1: Let i be the total number of MDPs seen so far
2: Let $i_{-j,c}$ be the number of MDPs assigned to class c after class j is removed for all j, c
3: Sample C_j according to:

$$\Pr(C_j = c) \propto \begin{cases} \frac{i_{-j,c}}{i-1+\alpha} F(\theta_c, M_i), & 1 \leq c \leq K \\ \frac{\alpha/m}{i-1+\alpha} F(\theta_c, M_i), & K+1 \leq c \leq K+m \end{cases}$$

the sample with highest posterior likelihood $P(M_i|\theta_c, O_i)$. To generate $\hat{M}_i$ for Line 5 of Algorithm 3, we sample a model from $P(M_i|\theta_c, O_i)$ (where c is the class with highest posterior likelihood). Our agent uses the hypothesis model $\hat{M}_i$ to select a new action by computing the optimal value function for the MDP with model $\hat{M}_i$. The agent then acts greedily with respect to this value function for a fixed number of steps (Lines 6 and 7 of Algorithm 3). The newly observed data points are added to O_i, a new MDP is generated from the posterior, a new policy is computed, and so on until the agent finds an optimal policy for the current task. After the optimal policy is found, the inner loop of Algorithm 3 returns the estimate $\hat{M}_i$.

3.6 Empirical Evaluation

We argue that our Hierarchical Bayesian MTRL algorithm will successfully learn a good estimate of the true prior distribution which will aid exploration in new MDPs. We argue the output for the posterior update, line 10 of Algorithm 3, is a reasonable approximation of the full posterior, and line 5 of Algorithm 3 will successfully use this information for exploration. Moreover, the algorithm will avoid negative transfer.

To test our hypotheses, we consider a simple colored maze domain inspired by the

foraging behaviors of rodents. Rodent foraging behavior is influenced by a variety of observable environment variables including chance of predation (scent or fur from predator species), quality of food sources (quantity of food), physical barriers (high gradient slopes between nest and food source), and available concealment (non-edible vegetation). Work by (Sullivan *et al.*, 2001) examined the impact of topography on the foraging behavior of heteromyid rodents (specifically the kangaroo rat Dipodomys spp.). Their observations suggest that the foraging behavior of the kangaroo rat is influenced by the costs of traversing terrain. Specifically, the data suggests that food resources at lower elevations are visited more frequently, due to lower costs of reaching the available seed deposits, but more time is spent exhausting resources at higher elevations in order to mitigate the costs of travel. Furthermore, the study states that high elevation feed sites were discovered later than low elevation feed sites. Due to natural seed density, which is highest at the bases of hills, it is less likely that food deposits will be found in higher elevation areas. Potentially, this explains the slower discovery of harder-to-reach artificial feed locations. Studies by (Mandelik *et al.*, 2003) and (Kotler *et al.*, 1994) examined the impact of predation risks on the foraging behavior of spiny mice (Acomys cahirinus) and gerbils (Gerbillus andersoni allenbyi) respectively. Specifically, the studies examined the path preferences of these foraging rodents when they were exposed to simulated predator sign. Both species show a preference for complex micro habitats which provide significant concealment during travel to food deposits, and also provide protection when foraging at food deposits. The relative costs of selected paths were influenced by artificially inserted predator sign which resulted in reduced foraging activity in the surrounding regions. Rodents appear to select foraging paths based on the costs of the traversed regions. Interestingly, both studies also note the global impact of lighting conditions on the exploratory behavior. A consequence of lighting conditions appears to be a global change

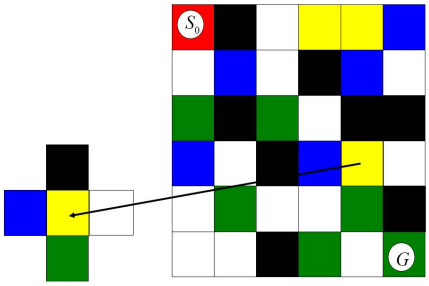


Figure 3.3: Example Colored Map. There are six types of locations. The agent begins in initial state s_0 and must travel to the goal location G. The cost of occupying a square is a function of the colors in the occupied square and adjacent squares (see the example square). To perform well the agent must learn the color costs and select the least cost path from start to finish.

in the relative costs of traversing habitat. For instance, while dark nights may cause the rodent to seek brush a bright moon can create long shadows near rocks providing significant camouflage for foraging rodents. This suggests that rodents take advantage of the perceived costs of regions and incorporate conditional global information influencing those costs when executing their foraging behaviors. Optimal foraging decomposes into separate task specific behaviors. Inspired by these studies we develop a simple colored maze domain which models some of the characteristics of foraging problems. However, our goal is not to provide an accurate model of rodent foraging. Instead we aim to abstract the foraging problem and show how knowledge transfer can improve the speed of learning optimal behaviors when the environment is subject to changes. Our agents will learn the costs of traversing regions of the domain, the locations of feed sites in the domain, and distinct classes of behaviors associated with global changes in the domain characteristics (including costs and feed site location).

A colored maze is 2-D planar surface divided into equal-sized squares. The agent

occupies one of these squares and can navigate to adjacent squares by executing movement actions. Each square of the domain is randomly colored with one of n colors representing micro-habitat in the area. Figure 3.3 illustrates an example colored maze. In our first experiments with this domain the agent must traverse randomly-generated colored mazes to a fixed goal location representing a food deposit. The objective is to maximize the total reward per episode by minimizing the path cost. Episodes end only after the goal is reached. In this domain, we are principally concerned with modeling the reward function, and presume the transition function is given. Thus, the agent is not initially aware of the costs associated with each micro-habitat. The parameters of the reward function w will characterize each task. Reward is a linear Gaussian function of the habitats surrounding the agent $r \sim Norm(wq, \sigma^2)$, with known variance σ^2. The color indicator vector q contains $5n$ bits. A component of the color vector q is set to 1 if the d^{th} adjacent square (current,up,down,left,right) is colored by the j^{th} color (only 5 bits are 1 at each time step all other bits are 0).

To frame this as a multi-task problem, we assume tasks are distinguished by their reward functions corresponding to global changes in the environment and reflected in task-specific distributions for color weights. We choose a Multivariate Gaussian prior over the color weights $w \sim Norm(\mu_c, \beta_c)$ to represent the within class distribution over MDPs. For the task prior G_0, representing a prior distribution for parameters (μ_c, β_c), we use the Normal Inverse Wishart (NIW) distribution. To construct tasks, we sample a small number of class distributions from the NIW defining the true set of underlying classes. Given the fixed set of true classes, we sample MDPs by first selecting a class by sampling from a multinomial distribution over the set of classes, and then sampling a task weight vector w_i from that class. A sequence of MDPs is drawn in this way.

Figure 3.4 shows initial experiments with 8 different colors and 4 classes. To evaluate

the agent's performance, we report the averaged total reward per episode on a fixed sample of 50 test MDPs. In this domain we see that the reward function is learned rapidly and so convergence to the optimal policy occurs early. Essentially every step the agent takes gives some information about the reward function. Despite these ideal conditions we observe in Figure 3.4 that the MTRL algorithm derives a benefit from transfer. After experience in 16 training MDPs the agent finds an optimal least cost solution in approximately 100 steps, which indicates that the estimate of the posterior closely approximates the true distribution over tasks. In contrast, more than 2500 steps are required for the uninformed algorithm to reach the same level of performance. The curves also indicate that the algorithm avoids negative transfer. Exploring according to the estimated posterior leads to consistently improved performance. We also observed that the inference procedure finds the true number of underlying classes.

In order to get a better sense of the benefit of our MTRL algorithm we constructed an environment where an uninformed prior is unlikely to do well. We modify the original domain so that the goal location may change from map to map. Such changes could represent the introduction of new feed sites within a habitat. The agent is returned to the starting state and the episode ends after finding the goal location which may be any square on the map. The objective is to maximize the total reward per episode. The map construction algorithm and distribution over the reward weight vectors remains unchanged. We add to this a distribution over goal locations. To capture uncertainty in the goal location we choose a discrete approximation of the Gaussian distribution over the (x, y) coordinates of the map. In this domain classes are summarized by the means and variances of these distributions. The mean represents the center of a class of feed locations, and the variance represents the width of the feed location distribution. From these class distributions a specific feed location is sampled and associated with a task. We

use the same component and hyper prior distributions is as described above. The class location distributions are gaussian and the class prior is a NIW distribution. To generate tasks we first jointly sample a class of weight vectors and a class of goal locations. A concrete task is generated by sampling from these class distributions yielding both a target weight vector and a specific goal location.

Results for maps of size 20x20 and 30x30 with 4 true underlying goal location classes (each underlying class is biased to have the goal location distributed around a fixed location of the map) are shown in Figure 3.5 and Figure 3.6. These graphs show the total reward received during the first episode after seeing i previous MDPs. Examining the performance on the first episode highlights the impact of the informed prior distribution which reduces the search problem to testing a relatively small number of locations corresponding to the learned classes. The horizontal lines in the graphs indicate the performances of the single task RL algorithm, which does not benefit from experience in previous tasks. We do not show within environment learning curves. In this domain, the improvement from learning the color values is negligible by comparison to the reduction in search time resulting from discovering the goal.

The observed improvement in average total reward during the first episode is dramatic. The cost of exploration is reduced by almost a full magnitude after ten tasks. We observed that the algorithm quickly discovers the small set of goal location classes, and then focuses exploration on areas around the class means. This intelligent search process is much more efficient than the nearly exhaustive search performed by the uninformed agent. This is particularly true in larger maps (Figure 3.6), and clearly demonstrates the value of our MTRL algorithm. The algorithm is also observed to avoid any significant level of negative transfer. The ability to propose new classes prevents the past task experience from overwhelming the information in the current observation set. In both

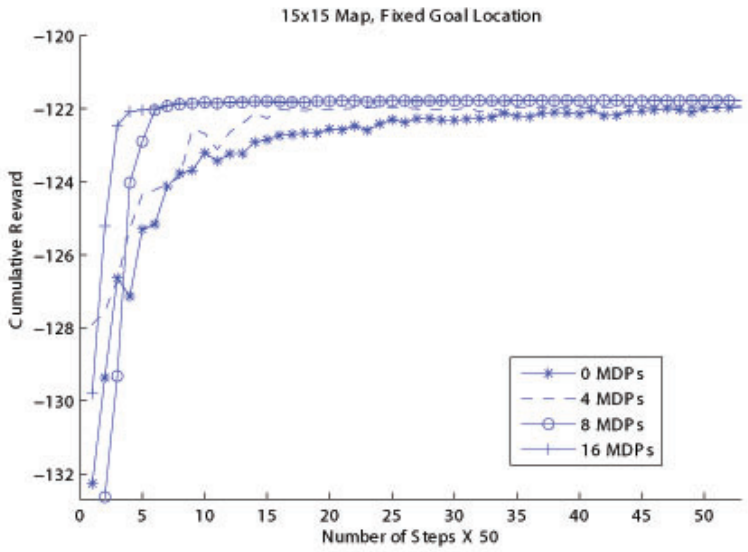


Figure 3.4: Domain 1: Average performance on the test set.

domains the algorithm finds a good estimate of the true underlying prior and successfully uses its estimate to improve exploration in new tasks.

3.7 Conclusion

In this chapter we have discussed the Bayesian representation of domain model knowledge for the MTRL problem setting. We have suggested that hierarchical Bayesian models provide an attractive framework for representing, learning, and reasoning about shared domain model knowledge. We introduced a framework for constructing hierarchical models representing classes of MDPs, an inference procedure for automatically learning the number and type of classes, and an algorithm for exploiting the posterior class structure when exploring new tasks. Our approach to modeling the class structure is based on an adaptation of Dirichlet Process priors. The DP formally represents uncertainty in the

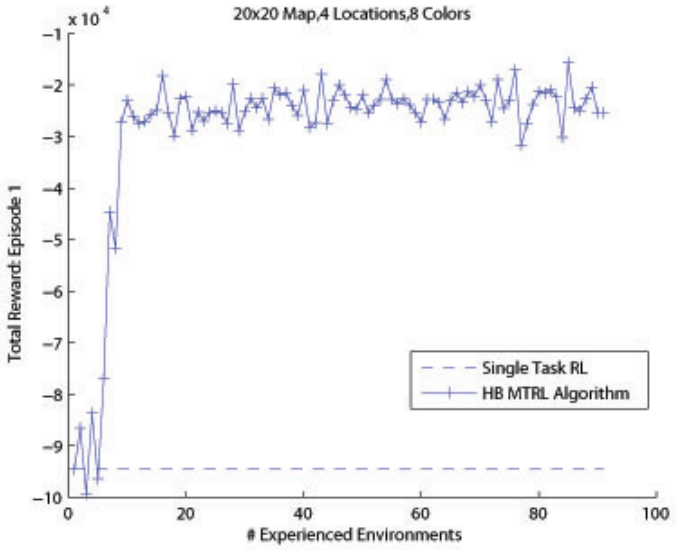


Figure 3.5: Domain 2: Performance on test MDPs plotted against the number of training MDPs. 20×20 map

number of classes and the class parameters. Our model extends the standard DP model by inserting an additional hierarchical layer. The standard DP prior decomposes observations. Our model decomposes the unobserved MDP model parameters. This allows our model to learn classes of MDPs and specific models for MDPs sampled from these classes. For purposes of posterior simulation we extend the algorithm presented in (Neal, 2000) to also sample MDPs. Our method of simulation automatically infers the class of new tasks and their model parameters. When existing classes, explaining the observations of earlier tasks, also explain the current task observations the agent benefits from transfer. Our action selection algorithm exploits the inference procedure by sampling a good model estimate from the posterior distribution, planning in the corresponding MDP, and acting optimally for a fixed number of steps. Intuitively, the action selection algorithm executes a sequence of tests to determine the class of the new MDP.

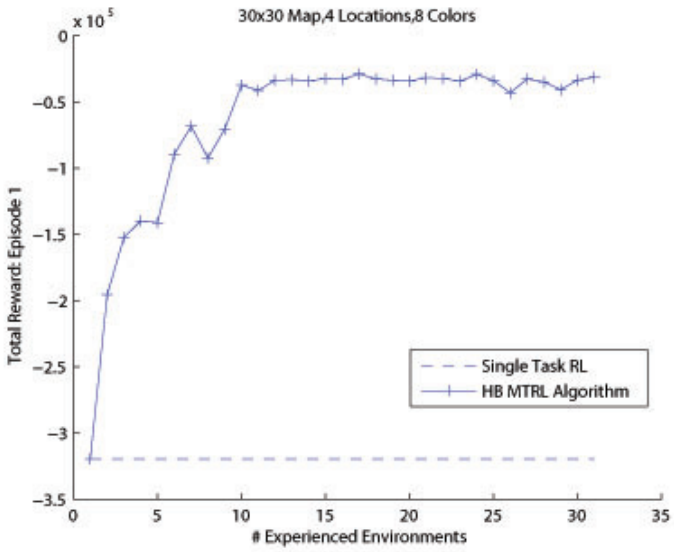


Figure 3.6: Domain 2: Performance on test MDPs plotted against the number of training MDPs. 30×30 map

We experimented in a colored grid-world domain inspired by experiments in animal foraging. We constructed classes of mazes distinguished by the costs of traversing grid sections and the locations of goal sites. A sequence of tasks were generated from an artificially constructed hierarchical class structure and presented to our MTRL agent. When tasks were distinguished by the travel costs alone the algorithm was able to learn the class structure and exploit this information to improve exploration. Given experience from as few as 8 tasks, the number of steps needed to identify the optimal policy was reduced by an order of magnitude. Additional experiments tested the benefit of learning the location of goal sites (feed sites). By learning the locations of food sites, the agent can execute a small number of tests to determine the goal location. We illustrated that our algorithm could successfully learn hierarchical distribution of goal sites. Furthermore, discovering the locations leads to a significant reduction in cost-to-goal during the first

episode. Intuitively, learning the hierarchical class structure reduces the number of experiments the agent must execute to determine the goal site.

Chapter 4: Hierarchical Bayesian Transfer via Policy Search

> When you watch an ant follow a tortuous path across a beach, you might say, "How complicated!" Well, the ant is just trying to go home, and it's got to climb over little sand dunes and around twigs. Its path is generally pointed toward its goal, and its maneuvers are simple, local responses to its environment. To simulate an ant, you don't have to simulate that wiggly path, just the way it responds to obstacles. - Herbert Simon

In this chapter we describe a Bayesian method of specifying policy knowledge transferrable across tasks. The previous chapter introduced a hierarchical Bayesian model for transfer of domain model knowledge across RL tasks. When new tasks share domain model structure transferred domain model knowledge improves the learning speed in the new domain. In the extreme case the domain model for the new task exactly matches the domain model of a previously observed task. In such cases the agent need only discover this relationship and can thereafter exploit the previously learned knowledge to plan its action selections. Anytime the agent can benefit from its knowledge it must plan, solve the MDP, to discover the optimal solution. This is a significant computational burden and it would be ideal if the agent could benefit from transferred knowledge without requiring this expensive step. Moreover the specification of domain models is not straightforward in many real world domains. Specifying the transition function for real world tasks is often its own research problem. Transferring of policy knowledge addresses both of these difficulties. The complexity of the solution space is often independent, and

much simpler, from the complexity of the domain. People often have insight into solutions to MDPs even if they cannot easily specify domain models. Policy knowledge is also directly transferrable to new tasks. Instead of indirectly specifying knowledge in the form of domain models, and then solving the MDP to discover the policy, the policy (or components of a policy) can be directly transferred. Below we motivate a particular problem domain useful for examining the problem of Bayesian policy transfer in RL.

In most real-world domains, there are multiple agents or agents that play different roles in jointly accomplishing a task. For example, in a military battle, a tank might engage the enemies on the ground while an attack aircraft provides the air cover. In a typical hospital, there are well-delineated roles for the receptionists, nurses, and the doctors. In this paper, we consider the general problem of discovering the roles of different agents and transferring that knowledge to accelerate learning in new tasks. The importance of roles in multi-agent reinforcement learning is well documented (Martinson & Arkin, 2003; Stone & Veloso, 1999; Marsella *et al.*, 1999). For example, (Marsella *et al.*, 1999) notes that individual agents in Robocup soccer tend to find policies suited to specific roles. Importantly, they note that sharing experiences between agents specialized in different roles can actually be detrimental to performance. Thus a key problem in multi-agent RL is to group the agents into different classes of roles and learn role-dependent policies for the agents.

We approach the role learning problem in a Bayesian way. In particular, we specify a non-parametric Bayesian prior (based on the Dirichlet Process) over multi-agent policies that is factored according to an underlying set of roles. We then apply policy search using this prior. Drawing on the work of (Hoffman *et al.*, 2007), our approach to the policy search problem is to reduce policy optimization to Bayesian inference by specifying a probability distribution proportional to the expected return which is then searched

using stochastic simulation. This approach is able to leverage both prior information about the policy structure and interactions with the task environment. We define such a distribution for our multi-agent task, and show how to sample role-based policies from it. Thus, our work can be viewed as an instance of Bayesian RL, where priors are learned and specified on multi-agent role-based policies.

Previous work on Bayesian RL has considered priors on the domain dynamics in model-based RL (Dearden *et al.*, 1999; Strens, 2000; Poupart *et al.*, 2006a; Wilson *et al.*, 2007) and priors on expected returns, action-value functions, and policy gradients in model-free RL (Dearden *et al.*, 1998; Engel *et al.*, 2005; Ghavamzadeh & Engel, 2007a). Our work is contrasted with these past efforts by being both model-free and by placing priors directly on the policy parameters. In our role-based learning formulation, this kind of policy prior is more natural, more readily available from humans, and is easily generalizable from experience in related tasks.

The Bayesian approach allows us to address a variety of role learning problems in a unified fashion. First, we demonstrate learning roles in the supervised setting where examples of optimal trajectories are provided by an expert. We then show that priors learned in one task can be transferred via Bayesian RL to a related task. Finally we show that roles can be discovered along with policies through Bayesian RL. All of our experiments are done in the real-time strategy (RTS) game of Wargus, focusing on tactical battle problems between groups of friendly and enemy agents.

In summary, the key contributions of this chapter are as follows: 1) Introducing a novel non-parametric Bayesian model over role-based multi-agent policies, 2) Introducing a novel model-free Bayesian RL algorithm that can accept priors over policy parameters, 3) An evaluation of the approach in the real-time strategy game of Wargus, showing its effectiveness for supervised learning, transfer, and pure RL.

4.1 Problem Formulation

In the previous chapter tasks were related by their domain models and no explicit decomposition of the domain models was exploited during the transfer process. In this chapter we assume that the solutions, policies, for tasks have reusable structure. We assume that policies decompose in some way and seek to learn the policy substructures which can be reused between tasks. For our illustrative example the multi-agent policy decomposes into a set of roles, and reuse is achieved when the discovered roles can be composed to solve new tasks. Here we formulate Multi-Agent MDPs and the basic structure of the multi-agent policy space.

Multi-Agent MDPs. In this chapter we consider a special class of MDPs, multi-agent MDPs (MMDPs), which model the problem of central control of a set of cooperative agents. An MMDP is simply a standard MDP where the action space and rewards are factored according to a set of m agents. In the MMDP setting the action space A is the product space $A_1 \times \ldots \times A_m$, where A_j is the action set of the j^{th} agent. The reward function R is similarly defined as the sum of individual agent rewards $R(s,a) = \sum_i R_i(s,a_i)$, where $R_i(s,a_i)$ is the reward received by agent i for taking action a_i in state s. The primary complicating factor of solving MMDPs compared to standard MDPs is that the action space is exponential in the number of agents. This means that the desired policy π is a probabilistic mapping from states to an exponentially large joint action space. The key questions are how to compactly represent such policies and learn them from experience.

Pseudo-Independent Policies. In general, large multi-agent domains like Wargus may require arbitrary coordination between agents. Unfortunately representing and learning coordination of this kind is computationally prohibitive for anything but very

small problems. To alleviate the computational burden we focus on learning a restricted class of joint agent policies, which in practice are often sufficient to achieve good performance. In particular, we use a pseudo-independent multi-agent policy representation where agents are assumed to be ordered and the decisions of agents at each time step are made in sequence, with later agents being allowed to condition on the decisions of previous agents. More formally, under our representation the probability of selecting joint action $A = (a_1, \ldots, a_m)$ in state s is,

$$P(A|s) = \prod_u P_u(a_u|s, A_{1,u-1}) \tag{4.1}$$

which is simply a product over individual agent policies P_u, where the policy for agent u is allowed to condition on the state s and the actions of previous agents in the ordering represented by $A_{1,u-1}$. This pseudo-independent representation allows for efficient representation and evaluation of a policy, at the expense of some expressive power in the allowed coordination structure.

Due to the large state and action spaces it is not possible to represent each agent policy P_u explicitly. Thus, we use a common parametric log-linear representation. For this purpose, we assume the availability of a feature function $g(s, a_u, A_{1,u-1})$ that returns a vector of numeric features for a state s, agent action a_u, and the actions of previously ordered agents. Given this feature function, each agent policy is represented via the following probability distribution,

$$P_u(a_u|s, A_{1,u-1}) = \frac{\exp(\theta_u \cdot g(s, a_u, A_{1,u-1}))}{\sum_{a'_u} \exp(\theta_u \cdot g(s, a'_u, A_{1,u-1}))} \tag{4.2}$$

where θ_u is the parameter vector for agent u. Together the set of parameters $\theta =$

$(\theta_1, \ldots, \theta_m)$ define a joint policy over agent actions.

Role-Based Parameter Sharing. In general, a pseudo-independent policy can specify different policy parameters for each agent. However, doing so does not exploit the fact that agents can often be divided into a small number of possible roles, where agents of the same role have identical policies. For example, in Wargus roles arise naturally because the agents, or units, have different characteristics which determine their suitability to particular tasks (e.g. the agents speed, range, durability, and power of their attack). We can capture such role structure by specifying the following: 1) the number of possible roles, 2) a set of policy parameters θ_c for each role c, and 3) for each agent u a role assignment function ϕ. Given this information the joint policy is simply a pseudo-independent policy with parameters $\theta = (\theta_{c_1}, \ldots, \theta_{c_m})$. The advantage of this representation is the reduction in the number of parameters that need to be learned. In particular, agents assigned to role c can benefit from the experiences of all other agents in the same role when learning the parameters θ_c. In practice a designer may not know the best role structure for a domain before learning begins, or even the number of roles that should exist. This motivates our goal of automatically learning and exploiting a prior over the role structure, including the number of roles, the role parameters, and the assignment of roles to agents.

We visualize the structure of this prior, accounting for transfer between a sequence of m tasks, in Figure 4.1. The parameters V_i represent the expected return of the role-based policy in task i. The observed return depends on the set of roles and the role assignment function which maps roles to units in the game. It is assumed that roles can be shared between tasks and that assignment functions are task specific.

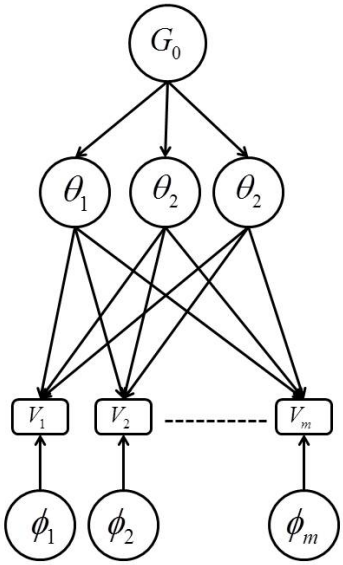


Figure 4.1: Illustration of the role-based policy structure. Roles θ_i are generated from the prior G_0. The role assignments for task j are governed by parameter ϕ_j. Learned roles are shared between tasks. The V_i represent the expected return given a role based policy for task i.

4.2 Bayesian Policy Search

Background. We formulate the MMDP learning problem as Bayesian policy search. In particular, we assume the availability of a prior distribution $P(\theta)$ over policy parameters, which might be learned from related problems or provided by a human. In the next section we will specify a particular form for this prior that captures uncertainty about the number and types of roles suitable for a given domain. Here we describe a novel model-free Bayesian policy-search algorithm that can exploit priors in order to speed up learning.

The approach taken in this thesis is motivated by work in the area of Bayesian optimal design. Optimal design problems focus on finding decision (policy) parameters θ that achieve maximum expected utility subject to our uncertainty about a domain. The uncertainty is specified via a prior distribution over the domain model parameters ω and the inherent stochasticity of the experimental process generating observations ξ, $P(\xi|\theta,\omega)$, which depends on both the domain model and decision parameters. The objective is to find the optimal parameters,

$$\theta^* = \arg\max_{\theta} U(\theta), \quad U(\theta) = \int U(\theta,\xi,\omega)P(\xi|\theta,\omega)P(\omega)d\omega d\xi \tag{4.3}$$

where $U(\theta,\xi,\omega)$ gives the utility of the decision θ given model parameters ω and observations ξ. In many cases, the integral does not have a closed form and Monte Carlo estimation is used for its evaluation.

Our basic approach differs in that we focus on priors placed directly on the decision space rather than the model parameters. This is an important distinction. The standard approach in Bayesian optimal design expresses a prior distribution over possible *worlds*.

It states a priori which world the agent is most likely to live in. Our framework places a prior distribution on the solution space, typically a much simpler object, and effectively states which solutions are most likely to be optimal. Therefore, the prior acts as a bias focusing probability mass on specific regions of the solution space.

For large decision spaces, exhaustive search for the optimizing parameters is infeasible and a number of optimization techniques have been studied. Our approach is inspired by (Muller, 1998) where the artificial distribution of Equation 4.4 is defined and then sampled from.

$$q(\theta, \xi, \omega) \propto U(\theta, \xi, \omega) P(\xi|\theta, \omega) P(\omega) \tag{4.4}$$

The construction of this artificial distribution assumes that the utility function is positive and bounded, which is typically easy to achieve in most settings. A key characteristic of this distribution is its marginal probability, marginalized over ξ and ω, which is proportional to $U(\theta)$. Due to the construction of q an algorithm designed to sample decision parameters from this distribution will tend to focus samples on parameters with high utility insuring that the most promising areas of the decision space are explored.

Specifically we build on (Hoffman *et al.*, 2007) which uses prior information on the policy parameters in a model based RL setting. Central to this work is the construction of an artificial distribution, proportional to a utility term and a prior on the policy parameters influencing the utility,

$$q(\theta) \propto U(\theta) P(\theta) = P(\theta) \int R(\xi) P(\xi|\theta) d\xi. \tag{4.5}$$

In the RL setting ξ corresponds to finite length trajectories from initial states to terminal states. The conditional distribution $P(\xi|\theta)$ is simply the probability that a policy

parameterized by θ generates trajectory ξ. And the utility is defined to be the expected return $E(\sum_{t=0}^{T} R(s_t, a_t)|\theta)$ for the policy parameters. Importantly, samples drawn from this artificial distribution focus on policies with high expected return.

Our approach. Given these definitions one can sample from the augmented distribution $q(\theta)$ to search for a policy maximizing the marginal $\theta^* = \arg\max_\theta U(\theta)P(\theta)$. Unfortunately sampling from q is not practical for our problems, because we do not have access to the domain model necessary for generating sample trajectories from $P(\xi|\theta)$. Moreover generating a large number of sample trajectories based on actual experience is costly. This motivates our adaptation of this approach to the model-free RL setting.

We elect to sample directly from an estimate $\hat{U}(\theta)P(\theta)$ of the target marginal distribution. To evaluate this product at any point θ we propose to use a finite sample drawn from a sequence of policies. To do so we use importance sampling. Importance sampling allows us to evaluate, in off-policy fashion, the expected return of any policy we wish. Importance sampling estimates the expected return using sets of trajectories generated by previously executed policies. The importance sampled estimate of the expected return is defined to be,

$$U(\theta) = \int P(\xi|\theta_i)\frac{P(\xi|\theta)}{P(\xi|\theta_i)}R(\xi)d\xi \approx \sum_{\xi} w(\xi)R(\xi) = \hat{U}(\theta)$$

The parameters θ_i indicate the policy parameters used to sample the corresponding trajectory and $w(\xi)$ is the weight $\frac{P(\xi|\theta)}{P(\xi|\theta_i)}$ of the sampled trajectory. In this way samples generated from other policies, the θ_i, can be used to estimate the returns of policy θ. Please see (Shelton, 2001) for further details on the weighted importance sampling algorithm used to estimate the expected return and its gradient in this paper.

Intuitively, our agent will alternate between stages of action and inference. It gener-

Algorithm 6 Bayesian Policy Search

1: Initialize parameters: $\theta_0, \xi = \emptyset$
2: Generate n trajectories from the domain using θ_0.
3: $\xi \leftarrow \xi \cup \{\xi_i\}_{i=1..n}$
4: $S = \emptyset$
5: **for** $t = 1 : T$ **do**
6: $\quad \theta_t \leftarrow Sample(\hat{U}(\theta_{t-1})P(\theta_{t-1}))$
7: $\quad$ Record the samples: $S \leftarrow S \cup (\theta_t)$
8: **end for**
9: Set $\theta_0 = argmax_{(\theta_t)\in S} U(\theta_t)$
10: Return to Line 2.

ates observed trajectories by acting in the environment according to its current policy, then performs inference for the optimal policy parameters given its experience so far, generates new trajectories given the revised policy parameters, and so on. An outline of our Bayesian Policy Search (BPS algorithm) procedure is given in Algorithm 6. This is a general method of exploiting a policy prior in the model free RL setting. The next section details the specific prior used in our MMDP setting and the Markov Chain Monte Carlo (MCMC) algorithm used to generate samples from $\hat{U}(\theta)P(\theta)$ (Line 6 of the BPS algorithm).

4.3 Role Learning with a DP prior

A Role Based Prior. To encode our uncertainty about the type of roles (what behaviors we expect), the number of roles, and the assignments of agents to roles we use a modification of the Dirichlet Process (DP) prior distribution which has all the properties we desire. Below we discuss the DP as a generalization of a standard finite mixture model and the modifications of the inference process we employ to sample role based policies.

The hierarchical prior distribution for the role parameters, which describes a joint policy for all agents and tasks, is shown in Figure 4.2(a). The prior illustrates how roles are combined and associated with units in the game. There are N tasks. Each task is composed of U units. Each unit is assigned a vector of role parameters θ_{c_u} indexed by c_u. The variable ξ_u represents a set of trajectories, a sequence of state-action observations, made by agent u when executing its policy. Assignment of units to roles depends on unit-specific features f_u and a vector of role assignment parameters ϕ. The vector of assignment parameters ϕ represent a strategy for assigning the collection of units to roles. An instance of the role assignment strategy is associated with each of the N tasks. The set of available roles θ_{c_u} is shared across all of the N tasks. Taken together, the set of parameters, $\Psi = (\theta, c, \phi, \alpha)$, represents a joint policy for all observed tasks. The DP specifies a distribution over all of these parameters. Thus, in our case, inference in the DP is a search through all possible partitions of agents into roles. Though we do not give a full treatment of the DP in this paper we do specify how we sample from the standard conditional distributions used in Gibbs sampling algorithms for the DP (Neal, 2000).

In the previous chapter the parameters θ_c represented classes of domain models whereas in this chapter these are discrete reusable roles. In the transfer setting the agent will have access to a pool of these roles. The model allows the agent to reuse existing roles and to posit new roles when the existing set is inadequate for a new task. The only other parameters which must be learned in a new task are ϕ which govern the assignment strategy.

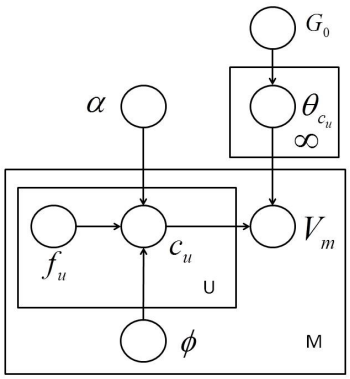


Figure 4.2: Plate notation for the hierarchical prior distribution for unit roles. The set of parameters Ψ includes: The prior distribution on role parameters G_0, the set of roles θ_c, the assignment parameters ϕ, unit features f_u, the concentration parameter of the DP model α, the assignment variable for each unit c_u, and the observed trajectory data ξ generated by executing role based policies. The plate notation indicates that there are N tasks and each task is composed of U units.

We define our own artificial probability distribution,

$$q(\theta, c, \phi) \propto [\widehat{U}(\theta)][\prod_j G_0(\theta_j)]P(c|\phi), \tag{4.6}$$

which decomposes into the estimated expected return $\widehat{U}(\theta)$, and the prior distribution $P(\theta, c) = [\prod_j G_0(\theta_j)]P(c|\phi)$. The prior further decomposes into two parts. The first factor is the prior over role parameters, $G_0(\theta_j)$, encoding our uncertainty about where we expect to find good solutions in the role parameter space. For our experiments we define G_0 to be a multivariate Gaussian distribution with zero mean and diagonal covariance. This asserts that a priori we believe the parameters of each role tend to be close to zero and that we do not know of correlations within the role parameter vector. The distribution $P(c|\phi)$ captures our uncertainty over role assignments. It is conditioned on a new set of parameters ϕ which influence the probability of assigning an agent to a

Algorithm 7 BPS For Role Based Policies

1: Initialize all parameters: (θ_0, c_0, ϕ)
2: Given the input ξ and the initialized parameters generate samples from $q(\theta, c, \phi)$.
3: $S = \emptyset$
4: **for** $t = 1 : T$ **do**
5: $\quad c_t \leftarrow SampleAssignments(\theta_{\mathbf{t-1}}, \mathbf{c_{t-1}}, \phi_{\mathbf{t-1}})$
6: $\quad \theta_t \leftarrow SampleRoleParameters(\theta_{\mathbf{t-1}}, \mathbf{c_t}, \phi_{\mathbf{t-1}})$
7: $\quad \phi_t \leftarrow SampleKernelParameters(\theta_{\mathbf{t}}, \mathbf{c_t}, \phi_{\mathbf{t-1}})$
8: $\quad S \leftarrow S \cup (\theta_{\mathbf{t}}, \mathbf{c_t}, \phi_{\mathbf{t}})$
9: **end for**

role as a function of the agent's features.

Overview: BPS for Role Based Policies. Algorithm 7 outlined here, is a specialization of line 6 of the BPS algorithm, designed to sample role based policies. At each point the state of the Markov chain will include a tuple of parameters, (θ, c, ϕ), which we will simulate in turn. The algorithm has three basic parts. First, after initializing the state of the chain, the new role assignments are sampled placing each agent into one of the existing components, or (using the mechanics of the DP) a new component generated from G_0. The second part of the algorithm updates the role parameters using Hybrid MCMC (Andrieu *et al.*, 2003b). The key to this portion of the algorithm is the use of log gradients to simulate samples from $q(\theta|c, \phi)$, focusing samples on regions of the policy space that are promising. Finally, we perform a simple Metropolis-Hastings update for the vector of kernel parameters. In the following sections we more explicitly define the implementation.

Sampling Role Assignments. The advantage of working with DP models is that they give rise to succinct conditional distributions for the assignment variable. In this chapter we exploit the DP conditional distribution to express the probability of role

assignments,

$$P(c_u = j | c_{(-u)}, \alpha) = \begin{cases} \frac{n_{(-u),j}}{n-1+\alpha} & |n_{(-u),j} > 0 \\ \frac{\alpha}{n-1+\alpha} & |\text{otherwise} \end{cases}, \tag{4.7}$$

encoding the probability of assigning agent u to role j given all the other role assignments $c_{(-u)}$. Note that the probability is proportional to the number of agents currently assigned to role j, $n_{(-u),j}$, and the DP focus parameter α (n is the total number of agents). The second term of this distribution assigns some probability to a role which currently has no data assigned to it allocating probability to unseen roles. This distribution can be used naturally to Gibbs sample new role assignments. Before we describe how this is done we modify the distribution.

As currently written the distribution depends only on the proportion of agents already assigned to a component. Ideally this distribution should take into account the similarities between agents when computing the conditional probability. To incorporate this idea we modify Equation 4.7, adapted from (Rasmussen & Ghahramani, 2002), to depend on the agent features by replacing $n_{-u,j}$ with the value of a parameterized kernel which we define to be,

$$n_{-u,j} = (n-1)\frac{\sum_{u' \neq u} K_\phi(f_u, f_{u'})\delta(c_{u'} = j)}{\sum_{u' \neq u} K_\phi(f_u, f_{u'})}. \tag{4.8}$$

Equation 4.8 determines the weighted sum of distances between the agent in question and all agents in role j and divides by the sum of distances between u and all other agents. We choose to use the kernel below which encodes a similarity metric between agent features, and allows the role assignment parameters to determine feature relevance,

$$K_\phi(f_u, f_{u'}) = exp(-\frac{1}{2}\sum_d (f_{u,d} - f_{u',d})^2/\phi_d^2). \tag{4.9}$$

We choose this kernel because it can decide when contextual features are irrelevant, and when differences in innate features such as 'range' strongly influence to which role an agent belongs. Principally the modified conditional distribution can substantially bias role assignments when the system is confronted with new tasks. This change was crucial to success in our experiments.

The assignment of agents to roles must be conditioned on their performance in the selected role. We encode this dependence in Equation 4.10. The equation has two primary factors. The first factor is new and encodes that the probability of assignment is proportional to the impact the role parameters have on the expected return $\widehat{U}(\theta)$. The second factor is the prior probability of the role parameters which we have already discussed. Due to our modification, simulation will tend to assign agents with similar agent features f_u to the same role.

$$P(c_u = j|c_{-(u)}, \theta, \phi, \alpha) = \begin{cases} b\frac{n_{-(u),j}}{n-1+\alpha}\widehat{U}(\theta)G_0(\theta_j)|n_{-u,j} > 0 \\ b\frac{\alpha}{n-1+\alpha}\widehat{U}(\theta)G_0(\theta_{new})|\text{otherwise} \end{cases} . \tag{4.10}$$

Importantly, we employ the same auxiliary sampling method described in the previous chapter to allow for new roles to be automatically generated by the prior. This means that we always store the set of roles which have agents assigned to them, and in addition store an auxiliary set of roles with parameters initialized from the prior G_0. In other words, the first case of Equation 4.10 corresponds to the existing roles and the second case governs the auxiliary roles. The Sample Assignments function samples a new assignment for each unit in turn exiting once all units have new assignments. In the next section we use hybrid MCMC to sample new role parameters.

Sampling Role Parameters. To update the role parameters θ we use Hybrid

Algorithm 8 Sample Assignments (θ, ξ, c)

1: Sample m auxiliary roles.
2: **for** each agent **do**
3: Sample new c_u according to Equation 4.10
4: **end for**
5: Return the new assignment.

MCMC (Andrieu *et al.*, 2003a; Duane *et al.*, 1987). Hybrid MCMC biases new samples using information about the log gradient of the target distribution. Doing so helps guide the sampling process for high dimensional distributions. In our case the target distribution, given the fixed assignments for all other parameters is $P(\theta|c,\phi) \propto \widehat{U}(\theta) \prod_j G_0(\theta_j)$. So long as we can compute the log gradient of this function we can use the hybrid rejection method. The expected return is estimated using importance sampling, and a treatment of the gradient we compute for the first term can be found in (Meuleau *et al.*, 2001). The second term is simply the log gradient of a product of multi-variate normal distributions which has a standard easily computable form. With the necessary gradient information in hand we can make use of Hybrid MCMC to construct a probing distribution for $P(\theta|c,\phi)$. Samples returned from Hybrid MCMC are used to update the state of θ.

Below, in the Sample Role Parameters function, Algorithm 9, we outline our Hybrid MCMC routine for optimizing the θ_j values given the assignment parameters. Generating new parameters occurs within the for loop. To bias the generated samples line 2 computes the vector of log gradients for role j. To construct the proposal we take a step proportional to the step size parameter ρ in the direction of the gradient and combine this value with Gaussian noise, d^*. The noise quantity perturbs the walk in the role parameter space allowing the algorithm to move out of local minima when large samples are used. Note that the acceptance probability, line 8, is proportional to both the ratio of

Algorithm 9 Sample Role Parameters (θ, ξ, c)

1: **for** $j \in Roles$ **do**
2: $x = \nabla \log(\widehat{U}(\theta) G_0(\theta_j))$
3: $\alpha \sim U_{0,1}$ and $d^* \sim N(0, I_{|\theta_j|})$
4: $d_0 = d^* + \frac{\rho}{2} \cdot x$
5: $\theta_{new} = \theta_j + d_0$
6: $d_1 = d_0 + \rho \cdot \nabla \log(\widehat{U}(\theta) G_0(\theta_{new}))$
7: $r = \frac{\hat{U}(\theta_{new}) G_0(\theta_{new})}{\hat{U}(\theta_j) G_0(\theta_j)}$
8: **if** $\alpha < A = \min(1, r \cdot exp(-\frac{1}{2}(d_1^T d_1 - d^{*T} d^*)))$ **then**
9: $\theta_j = \theta_{new}$
10: **else**
11: $\theta_j = \theta_j$
12: **end if**
13: **end for**
14: return θ_j

utilities and the ratio of prior distributions (the ratio is computed in line 7). The term in the exponential simply prevents the gradient values from being too large by comparison to the Gaussian noise. If the gradients are informative the proposal distribution tends to sample parameters which increase our estimate of the expected return; the sampling procedure will focus samples on regions of the policy space that appear fruitful.

Sampling Kernel Parameters. We use Metropolis-Hastings to update the kernel parameters given the sampled state for (c, θ), using a Gaussian proposal distribution. Here we use the pseudo-likelihood, proportional to $[\widehat{U}(\theta)][\prod_j G_0(\theta_j)][\prod_u P(c_u|\phi)]$, to determine acceptance.

Acting in the Environment. After drawing a large sample using this simulation procedure we wish to select a policy, a tuple of parameters (θ_t, c_t, ϕ_t), to execute in our task. To do so we select the policy maximizing,

$$(\theta^*, c^*, \phi^*) = argmax_{(\theta_t, c_t, \phi_t) \in S} U(\theta_t). \tag{4.11}$$

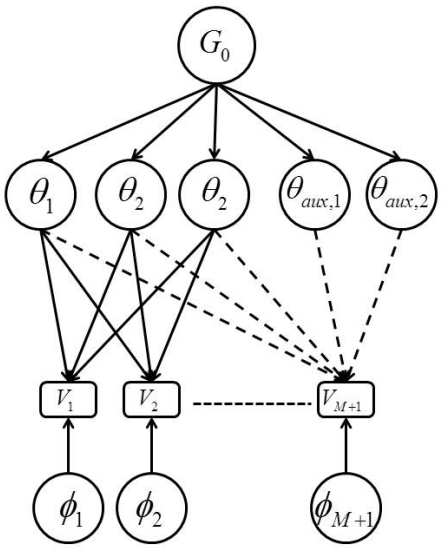


Figure 4.3: The BPS algorithm for Role Based Policies searches for a new assignment function mapping roles to units. New roles are selected from an auxiliary set when needed.

The sampled policy is then executed in the domain to generate new trajectories (Line 2 of the BPS algorithm).

4.4 Transfer of Policy Knowledge

Given M previously observed tasks, we take the following strategy. We fix the set of previously identified roles and use this role set as prior knowledge in the new task. Discovery of roles is a difficult process. Therefore, role reuse in the new task can dramatically improve learning speed. Within a new task we search for strategies that recombine these existing roles and propose new roles to overcome novel obstacles (Figure 4.3). The BPS algorithm is modified to implement this strategy by sampling role-based policies

given the fixed set of transferred roles. Implementing these changes is simple. The state (θ, c, ϕ) of the markov chain always includes the transferred roles and never modifies the role assignments for earlier tasks. The algorithm only modifies role assignments for units in the current task. As shown in Figure 4.3 the algorithm can still propose new roles when necessary.

4.5 Empirical Evaluation

We evaluate our algorithm on problems from the game of Wargus. RTS games involve controlling multiple agents in activities such as: resource gathering, building military infrastructure/forces, and engaging in tactical battles. We focus on tactical battles where we must control a set of friendly agents to destroy a set of enemy buildings and agents controlled by the native Wargus AI. Examples of Wargus maps are shown in Figure 4.4. Our learning agent controls the red units and directs them to destroy the opposing blue forces. In these examples a sub-goal of the the overall task is to keep the long ranged siege unit alive long enough to destroy the towers defending the lumber-mill. Example units include archers (ranged unit), the ballista (ranged unit), and knights (melee unit). Each unit must fulfill a specific task to overcome the opposition. For instance, the archers can kill knights before they reach the ballista, and the ballista, due to its long range, is ideal for sieging the powerful tower structures. These roles must be discovered and executed simultaneously to successfully overcome the opposition.

Decision cycles occur at fixed intervals, where all agents select an action. The possible actions include attacking any of the friendly or enemy units, or doing nothing. Attack actions cause an agent to pursue the selected target and attack when in range. At the end of each decision cycle the state is updated and the agents receive rewards, which include

Figure 4.4: Examples of Wargus conflicts. Each side is composed of a heterogenous collection of units and must destroy the other group (including structures).

a small reward for damaging a target plus a bonus reward for killing the target plus a large reward if the team has won the game (rewards are negative when attacking and killing friendly units). A game continues until one side is destroyed, or for a maximum of 100 decision epochs.

Our learning algorithm requires us to provide agent specific features f_u to facilitate role assignment. For this purpose, we use as features the basic attributes of agents as described in the Wargus configuration file including information such as armor strength, speed, and attack range. We also include features that capture properties of an agent's initial state such as distance between agent and enemy targets, and distance to friendly agents. Thus f_u captures both inherent physical capabilities of an agent as well as contextual information present in the state. Our log-linear policy representation also requires that we provide a feature set $g(s, a_u, A_{1,u-1})$ that captures properties of the current state s, candidate agent action a_u, and other agent actions. We hand-coded a set of 20 features that capture information such as whether the range of the target exceeds that of the agent, whether the target is mobile, and how many agents are currently

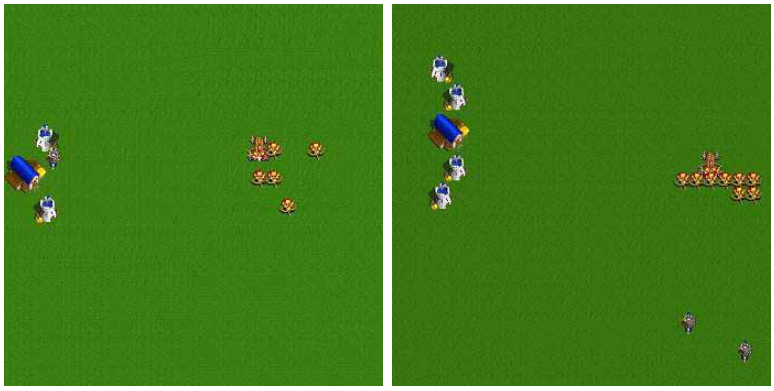
	Friendly			Enemy		
	Archers	Ballista	Knights	Towers	Ballista	Knights
Map 1	5	1	0	2	0	1
Map 2	5	1	1	2	1	1
Map 3	16	1	2	4	2	5
Map 4	7	1	0	2	0	2
Map 5	8	1	1	4	1	2

Table 4.1: Team compositions.

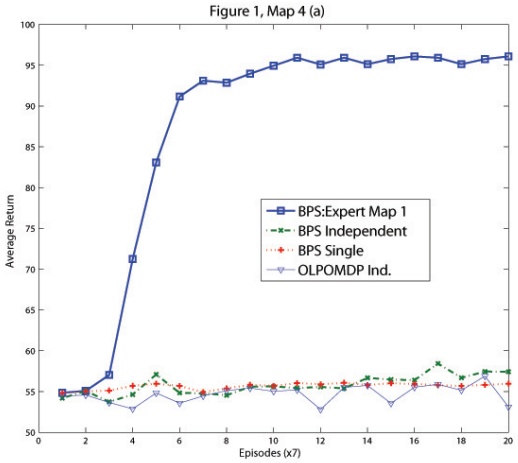
attacking the target.

We test our algorithm on a variety of Wargus maps with varying locations and compositions of units. The basic compositions of units for each map are shown in Table 4.5.

Learning from an Expert. An important advantage of our learning approach is that it naturally benefits from expert examples. Here we test the ability of our algorithm to discover the inherent role structure observed in expert play in Map 1, Map 2, and a much harder map Map 3. In each case, the algorithm is provided with 40 episodes of observed performance of high-quality (expertly coded) policies, which were based on natural role structures for each task. To test the performance of our algorithm we allow BPS to learn a policy given the expert samples as input. The performance of the learned policy is then assessed on the target maps. In all three cases the agents win 100 percent of games after learning (prior to learning no games are won). We assess how well the discovered roles match those used in the expert policies by examining the decomposition of units in the sampled assignment vector c^*. In each case BPS learns role structures consistent with the expert policy, including the appropriate role assignments and role parameters. Two roles are found for the first map, which correspond to archers defending the ballista and the ballista attacking the enemy base. The algorithm discovers that the knight deserves its own role in map 2 (it is good at killing the enemy ballista). Likewise, in the difficult third map the sampled policies partition the agents appropriately.



(a) Training Map: Map 1 (b) Evaluation Map: Map 4



(c) Results averaged over 20 runs

Figure 4.5: Results of RL after learning prior from expert play on Map 1.

Prior Transfer. Given expert demonstrations our algorithm is able to learn a role structure, role assignment function, and role parameters. We now evaluate the ability of BPS to transfer this learned knowledge, in the form of a prior distribution over parameters (θ, ϕ), to speedup RL on new, but related, tasks. In particular, we conduct two experiments. The first uses the posterior distribution learned from expert demonstrations on Map 1 as a prior for RL in Map 4. The second experiment is identical but uses the posterior learned from expert demonstrations in Map 2 as a prior when learning a policy for Map 5. We compare the transfer performance to three baselines. The first two baselines (BPS Independent and BPS Single) assume a pre-specified role structure and use the BPS algorithm to identify the role parameters. Recall that BPS for Role Based Policies searches for both a decomposition of agents into roles, for the parameters of the assignment function, and for the parameters defining each role. These baselines assume a fixed and known number roles and a fixed assignment strategy. Only the role parameters must be learned. The Independent baseline assigns all agents to their own role, and the Single baseline assigns all agents to a single role. Given these fixed role structures the BPS algorithms is used to update the role parameter values as described above. In principle BPS with the independent role structure can learn the correct policy and this fixed structure may be beneficial if the role structure is unimportant. Similarly, if a single role will suffice, then BPS run with all agents assigned to a single policy will be optimal. Because the Independent and Single baselines cannot make use of the role structure they cannot benefit from transfer. Instead they must learn their policies from scratch. Furthermore, we compare our performance to OLPOMDP where agents independently learn their own log-linear action selection policies (Baxter *et al.*, 2001). In principle OLPOMDP should be able to find a locally optimal policy.

The results for the first experiment are shown in Figure 4.5(c). The curve (BPS:Expert

Map 1) illustrates the advantages of transfer using the prior distribution learned from expert samples on Map 1. The graph shows the average reward per episode for the learned policy as a function of the amount of experience (number of episodes) used to learn the policy. The algorithm benefits substantially from the improved prior distribution as it focuses samples on policies with good role structures. We see similar results for the more complex Map 5 illustrated by the curve (BPS:Expert Map 2) in Figure 4.6(c). In this case a larger number of roles needs to be correctly mapped to new agents making the problem substantially more difficult when roles are not transferred. Our learning algorithm, with the improved prior distribution, substantially outperforms the baselines. It is worth noting the performance of BPS Single in Figure 4.6(c). On this map the baseline has discovered a simple but suboptimal policy accumulating a small amount of reward by sending all agents to attack a tower. This results in damaging the tower, but insures that all friendly units will die. The restricted policy representation, coupled with the large number of friendly agents, helps this baseline find this suboptimal policy quickly. The baselines were unable to find the optimal policy before we terminated the experiments.

Autonomous Role Discovery. Learning in both of the test maps becomes much more difficult when examples of expert play are not provided. This can be seen by the poor performance of the baselines, which cannot benefit from prior role knowledge. Here we consider whether our algorithm is able to start from an uninformative prior and autonomously learn the role structure during the actual RL process, and benefit from that structure as it is being learned. The BPS curve in Figure 4.7(b) shows results of our agent learning to win tactical conflicts without the benefit of prior experience in related tasks. Consequently, the agents prior is less informative. Given the less informed prior many more episodes are required before an optimal policy is found. However, the

algorithm does eventually learning a winning policy. This is a substantial benefit over the baseline algorithms which do not win games when provided with the same amount of experience. This shows that learning the role structure during the actual RL process can speed up RL without the transfer of previously discovered roles. The primary reason for this is that experience between agents of the same role is shared leading to faster learning of role policies once good role assignments are discovered. Qualitatively the algorithm did find distinct and natural roles for the archers and ballista. Results on Map 5 are not good for any of the non-transfer algorithms. We conjecture that it would require any RL agent without significant prior knowledge a very long time to discover a winning policy. In fact, when run with a random policy no win was recorded after 500 games. As shown in Figure 4.8(c) BPS without transfer performs no better than the baselines on this problem. However, as illustrated before, BPS can successfully make use of learned priors to significantly speed learning in new tasks. To take advantage of this property we first allow BPS to learn a useful role structure in Map 1 and then make use of this knowledge in Map 5. The results are shown in Figure 4.8(c), BPS:BPS Map 1; the prior learned in the simpler problem make the harder problem tractable. It is a substantial advantage of the BPS algorithm that it can naturally leverage experience in simple tasks to learn quickly in hard tasks.

4.6 Conclusion

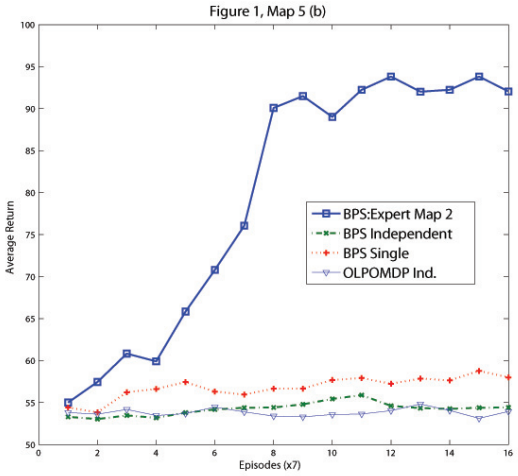
In this chapter we introduced a Bayesian method of expressing policy knowledge and an algorithm for exploiting that knowledge when searching for an optimal policy. The algorithm, based on stochastic simulation methods, uses the prior distribution to bias the search in policy space. The algorithm accepts any policy prior, but this chapter focused

on a particular hierarchical policy space. The important characteristic of this hierarchical policy space was a decomposition of the joint policy into a set of transferrable components. In the context of our experiments in the multi-agent Wargus game these components represented independent roles filled by distinct units in the game. The set of discovered components/roles coupled with an assignment function, mapping components to units, composed the complete hierarchical policy. Our policy search algorithm learned the complete hierarchical policy including the set of components, and a distinct assignment function for each task, inventing new components when needed. Our experiments illustrated that learned components could be successfully transferred leading to learning in increasingly difficult Wargus confrontations. In addition, we showed how our approach could use expert demonstrations for both learning and transfer. The policy search algorithm accepts trajectories from any policy including expert policies. When provided with expertly generated trajectories the agent quickly learns to match the experts performance. Notably, if the expert's performance is sub-optimal the policy search algorithm can discover this fact and find a better performing policy.

The BPS algorithm has two principle flaws. First, we interpret the prior distribution as stating $P(\theta = \theta^*)$ the probability that a given policy is in fact the optimal policy. This interpretation suggests a particular behavior for the posterior distribution: We expect zero mass on all policies with sub-optimal returns. As currently defined the posterior distribution, proportional to Equation 4.4, does not have this property. This artificial distribution is proportional to the expected return. As data accumulates the estimates of policy returns improves, but the posterior still allocates positive mass to sub-optimal policies. In this sense the expected return is not the ideal candidate for the utility function. In the following chapters we will discuss a utility function that possesses the necessary properties. When incorporated into the BPS algorithm it will

place zero mass on sub-optimal policies. Second, unlike most Bayesian approaches to RL the BPS algorithm does not address the problem of exploration. The selection of policies is determined by the estimated returns and does not directly incorporate the posterior uncertainty. We leave this issue to be resolved in future work, but we believe that the utility function discussed in the following chapters can directly resolve this problem.

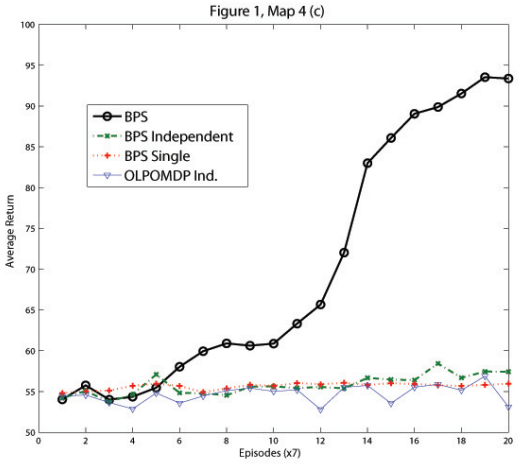
(a) Training Map: Map 2 (b) Evaluation Map: Map 5



(c)

Figure 4.6: Results for Map 5. Results of RL after learning prior from expert play on Map 2 (average over 20 runs).

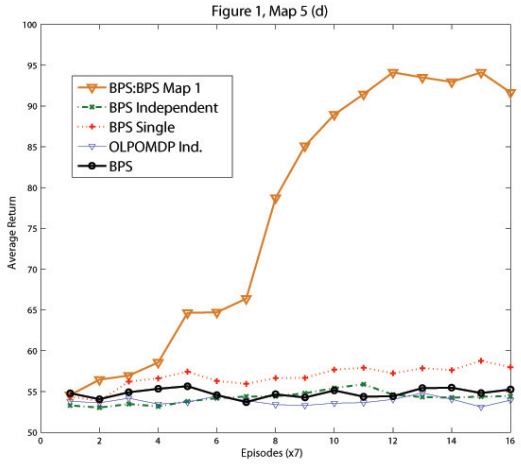
(a) Evaluation Map: Map 4



(b) Results averaged over 20 runs

Figure 4.7: Results for Map 4. Learning roles using RL.

(a) Training Map: Map 1 (b) Evaluation Map: Map 5



(c)

Figure 4.8: Results for Map 4 and Map 5. Learning roles using RL. Each graph shows the average total reward per episode (average over 20 runs).

Chapter 5: Bayesian Optimization for RL

In the policy search setting, RL agents seek an optimal policy within a fixed set. The agent iteratively selects new policies, executes selected policies, and estimates policy performance. Naturally, future policy selection decisions should benefit from the information generated by all previous selections. A question arises regarding how the performance of untried policies can be estimated using this data, and how to use the estimated performances to direct the selection of a new policy. Ideally, the process of selection accounts for the agent's uncertainty, and directs the agent to explore new parts of the policy space when uncertainty is sufficiently high.

This chapter introduces a method of policy search based on Bayesian Optimization (BO). BO is a method of planning a sequence of queries from an unknown objective function for the purpose of seeking the maximum. It is an ideal method for tackling the basic problem of policy search; BO directly confronts the fundamental conflict between exploration (global search) and exploitation (local search). Fundamental to application of Bayesian Optimization techniques is the definition of a Bayesian prior distribution for the objective function. The method of BO expends computational resources querying the surrogate representation of the objective function. Ideally, by using a large number of surrogate function evaluations the true maximum point can be identified with fewer queries of the true objective.

Success of BO relies on the quality of the modeling effort. In this chapter we focus on GP models of the expected return. The generalization performance of Gaussian Process (GP) models, and hence the performance of the BO technique, is strongly impacted by

the definition of both the GP mean function, and the kernel function encoding relatedness between points in the function space.

Prior work applying BO to the RL problem tackled difficult problems of gait optimization and vehicle control ((Lizotte *et al.*, 2007; Lizotte, 2008a; Brochu *et al.*, 2009)), but ignored the sequential nature of the decision process. Execution of a policy generates a trajectory represented by a sequence of action/observation/reward tuples. Previously, this information was reduced to a Monte-Carlo estimate of the expected return and all other information present in the observed trajectories was discarded. By taking advantage of the sequential data, we argue, and empirically demonstrate, that our methods dramatically improve the data efficiency of BO methods for RL. To take advantage of trajectory information we propose two complementary methods. First, we discuss a new kernel function which uses trajectory information to measure the relatedness of policies. Second, we discuss the incorporation of approximate domain models into the basic BO framework.

Our first contribution, is a notion of relatedness tailored for the RL context. Past work has used simple kernels to relate policy parameters. For instance, squared exponential kernels were used in (Lizotte *et al.*, 2007; Lizotte, 2008a; Brochu *et al.*, 2009). These kernels relate policies by differences in policy parameter values. We propose that policies are better related by their *behavior* rather than their parameters. We use a simple definition of policy behavior, and motivate an information-theoretic measure of policy similarity. Additionally, we show that the measure of similarity can be estimated without learning transition and reward functions.

Our second contribution, incorporates learned domain models (the transition and reward function) into the BO framework. Learned domain models are used to simulate Monte-Carlo policy roll-outs for the purpose of estimating policy returns. Crucially,

we consider the setting where the simulator *is not* a replacement for the true domain. The domain model class may have significant bias that prevents close approximation of the true transition and reward functions. Consequently, Monte-Carlo simulations of the environment using the learned functions can produce substantial errors that prevent the direct application of standard model-based RL algorithms. To overcome this problem, we propose using the GP model to correct for errors introduced by the poor domain model approximations, and show empirically that our algorithm successfully uses the learned transition and reward models to quickly identify high quality policies.

In the following sections we discuss the general problem of BO, motivate our modeling efforts, and discuss how to incorporate our changes into BO algorithms. We conclude with an empirical evaluation of the new algorithms on five benchmark RL domains.

5.1 Bayesian Optimization

Bayesian optimization addresses the general problem of identifying the maximum of a real valued objective function,

$$\theta^* = \arg\max_{\theta} \eta(\theta).$$

This problem could be solved by optimizing the objective function directly. For instance, the Covariance Matrix Adaptation algorithm (Hansen, 2006), which has already shown some promise in RL, directly searches the objective function by executing thousands of queries to identify the maximum. If each evaluation of the objective has a small cost, then thousands of evaluations could easily be performed. However, when individual evaluations of the objective incur high costs, algorithms which rely on many evaluations

are inappropriate. Examples of domains with this property are easy to find. For example, the evaluation of airfoil design and engine components rely on running expensive finite element simulations of gas flow. These simulations have extreme time costs; evaluations of each design can take upwards of 24 hours. Other domains, familiar to RL researchers, include robot control. Running robots is time-consuming and increases the likelihood of physical failures. In cases like these it is best to minimize the number of objective function evaluations. This is the ideal setting for the application of BO. To reduce the number of function evaluations the BO approach uses a Bayesian prior model of the objective function and exploits this model to plan a sequence of objective function queries. Essentially, BO algorithms trade computational resources to improve the quality of selected query points.

Modeling the objective is a standard strategy in learning problems where the true function may be approximated by e.g. regression trees, neural networks, polynomials, and other structures that match properties of the target function. Using the parlance of RL, the Bayesian prior model of the objective function, sometimes called the surrogate function, can be viewed as a function approximator. As a Bayesian prior the model has the additional property of expressing explicit uncertainty of the target function, and supports Bayesian methods of analysis. However, the model of the objective function has additional responsibilities in the BO framework. Unlike the standard application of function approximators in RL, which estimate the returns of a fixed policy, the objective function model in BO represents the complete surface of the expected return. Where standard function approximators generalize across states the models in BO algorithms generalize across policies, or state-policy pairs. This is necessary to support intelligently querying the surrogate representation of the objective function.

The BO method plans a sequence of queries. The process proceeds as follows: (1) A

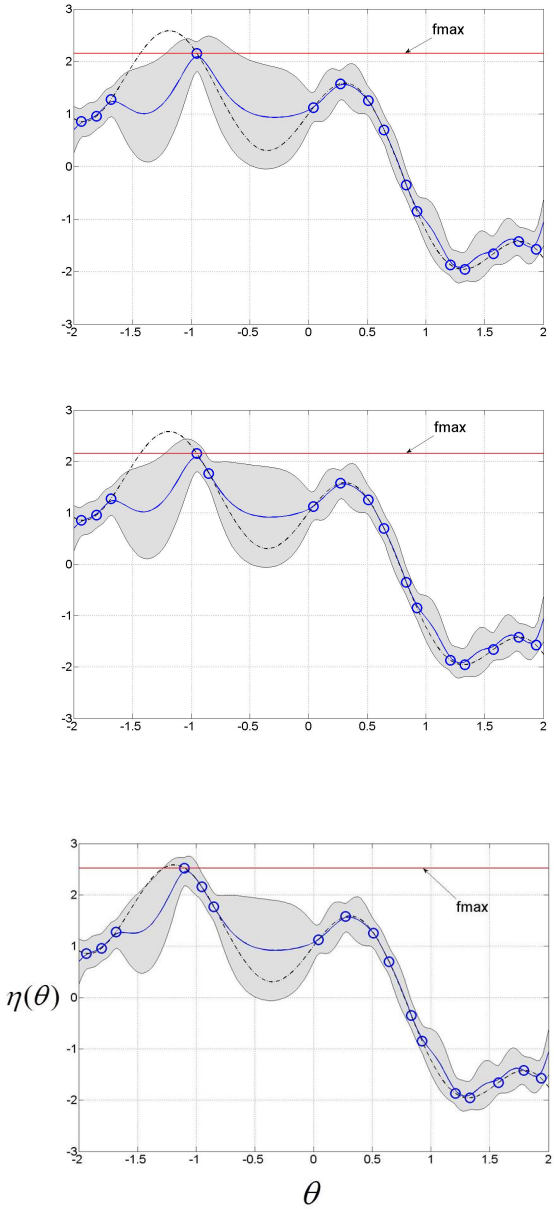


Figure 5.1: An illustration of Bayesian Optimization where a sequence of queries are made. The agent observes the objective function (dashed line) at a finite set of points (blue circles). Conditioned on the observations the agent maintains a model of the underlying objective function. The solid blue line depicts the mean of this model and the shaded regions illustrate the model uncertainty (2 standard deviations). Uncertainty is lower near densely sampled regions of θ space. The agent selects new data points for purposes of identifying the true maximum. As new observations are added the quality of the model improves and observations are focused near the maximal value.

query is selected. Queries are selected using a fixed criteria which considers the value of global and local explorations. (2) The objective function is observed at the selected query point by gathering real data from the system being optimized. Ideally, the computational resources expended in the previous step improves the quality of the observed data. (3) The system observes the performance at the query point and updates the posterior model of the objective function. (4) The process returns to step 1. This basic process is depicted in Figure 5.1. Below we discuss the key components of BO algorithms including the improvement function and the prior model of the objective function.

5.1.1 Query Selection

The selection criteria plays an important role in the quality of the exploration, and consequently the speed of identifying the optimal point. The basic problem of selection can be framed as identifying the point that minimizes the agent's expected risk,

$$\min_{\theta} \int \parallel \eta(\theta) - \eta(\theta^*) \parallel dP(\eta|D_{1:n}),$$

with respect to the posterior distribution (By minimizing the expected difference between $\eta(\theta)$ and the maximum $\eta(\theta^*) = \arg\max_\theta \eta(\theta)$ we maximize the value of $\eta(\theta)$). The data $D_{1:n}$ is a collection of pairs $D_{1:n} = \{\langle \theta_i, \eta(\theta_i) \rangle\}|_{i=1}^n$. Each pair is a previously selected policy point θ_i and the evaluated performance at that point $\eta(\theta_i)$. The posterior distribution $P(\eta|D_{1:n})$ encodes all of the agents knowledge of the objective function. This risk functional is a natural foundation for a myopic iterative selection criteria,

$$\theta_{n+1} = \arg\min_{\theta} \int \parallel \eta(\theta) - \eta(\theta^*) \parallel dP(\eta|D_{1:n}).$$

Unfortunately, this selection criteria requires solving a computationally demanding minimax problem. A heuristic method of selection must be used.

A common heuristic called Maximum Expected Improvement (MEI) ((Mockus, 1994)) is the method of selection used in this work. The MEI heuristic compares new points to the point with highest observed return in the dataset. We denote the value at this empirically maximal point to be η_{max}. Using this maximal value one can construct an *improvement* function,

$$I(\theta) = max\{0, \eta(\theta) - \eta_{max}\},$$

which is positive when $\eta(\theta)$ exceeds the current maximum and zero at all other points. The MEI criteria searches for the maximum of the expected improvement,

$$\theta_{n+1} = argmax_{\theta} E_{P(\eta|D_{1:n})}\left[I(\theta)\right]. \tag{5.1}$$

Crucially, the expected improvement function exploits the posterior uncertainty. If the mean value at a new point is less than η_{max} the value of the Expected Improvement may still be greater than zero. Consider the case where the posterior distribution, $P(\eta(\theta)|D_{1:n})$, has probability mass on values exceeding η_{max}. In this case, the expected improvement will be positive. An example is shown in Figure 5.2. Therefore, the agent will explore until it is sufficiently certain that no other policy will improve on the best policy in the data set. Due to the empirical success of the MEI criterion it has become the standard choice in most work on BO.

When the posterior distribution is Gaussian then the expected improvement function

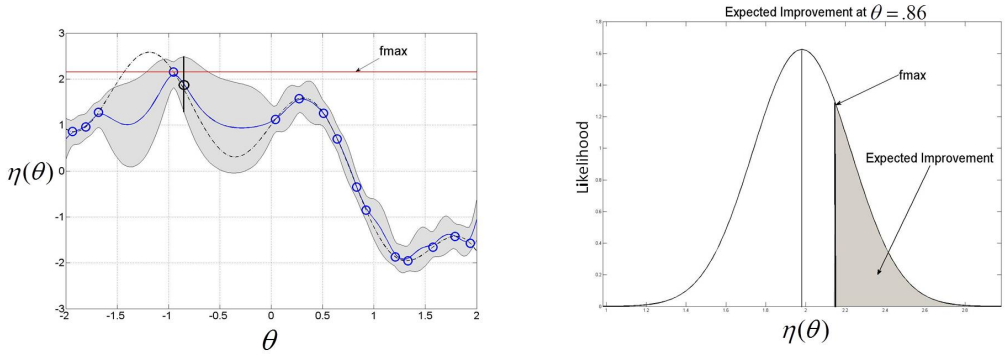


Figure 5.2: The expected improvement heuristic. The heuristic assesses the value of observing new points. On the left we consider the point circled (in black) at $\theta = .86$. On the right we illustrate the expected improvement assuming that $P(\eta(\theta)|\theta = .86)$ is Gaussian. To be considered an improvement the value of $\eta(\theta)$ must exceed the value of the current maximum $fmax$. The probability mass associated with the expected improvement is shaded. The expected improvement is proportional to this weighted mass.

has a convenient closed form solution,

$$E_{P(\eta(\theta)|D_{1:n})}\left[I(\theta)\right] = \sigma(\theta)[\frac{\mu(\theta) - \eta_{max}}{\sigma(\theta)}\Phi(\frac{\mu(\theta) - \eta_{max}}{\sigma(\theta)}) + \phi(\frac{\mu(\theta) - \eta_{max}}{\sigma(\theta)})].$$

The functions $\mu(\theta)$ and $\sigma(\theta)$ are the mean and standard deviation of the Gaussian distribution. Function $\Phi(.)$ is the cumulative distribution function of the standard Gaussian distribution, and $\phi(.)$ is the probability density function. Please note that the expected improvement function is zero when the standard deviation is zero.

5.1.2 Objective Function Model

As a Bayesian method the performance of BO depends profoundly on the quality of the modeling effort. The specification of the prior distribution determines the nature of the

posterior and hence the generalization performance of the surrogate representation. We elect to model the objective function using a Gaussian Process ((Rasmussen & Williams, 2005)),

$$\eta(\theta) \sim GP(m(\theta), k(\theta, \theta')).$$

GP models are defined by a mean function $m(\theta)$ and covariance function $k(\theta, \theta')$. The mean function specifies the expected value at a given point $m(\theta) = E[\eta(\theta)]$. Likewise, the covariance function estimates the covariance $k(\theta, \theta') = E[(\eta(\theta) - m(\theta))(\eta(\theta') - m(\theta'))]$ The kernel function encodes how correlated values of the objective are at points θ and θ'. Both of these functions encode knowledge of the underlying class of functions.

For the purpose of computing the improvement function described above, the posterior distribution at new points must be computed. In the GP model, this posterior has a simple form. Given the data $D_{1:n}$ the conditional posterior distribution is Gaussian with mean,

$$\mu(\eta(\theta_{n+1})|D_{1:n}) = m(\theta_{n+1}) - \mathbf{k}(\theta_{n+1}, \theta)\mathbf{K}(\theta, \theta)^{-1}\mathbf{y},$$

and variance,

$$\sigma^2(\eta(\theta_{n+1})|D_{1:n}) = k(\theta_{n+1}, \theta_{n+1}) - \mathbf{k}(\theta_{n+1}, \theta)^t\mathbf{K}(\theta, \theta)^{-1}\mathbf{k}(\theta, \theta_{n+1}).$$

Define $\mathbf{y}$ to be the column vector of observed performances such that $y_i = \eta(\theta_i)$ and $\mathbf{K}(\theta, \theta)$ to be the covariance matrix with elements $\mathbf{K}_{i,j} = k(\theta_i, \theta_j)$. Define $\mathbf{k}(\theta_{n+1}, \theta)$ to be the a column vector of correlations such that the i^{th} element is $k(\theta_{n+1}, \theta_i)$ ($\mathbf{k}(\theta_{n+1}, \theta)^t$ is the transpose of this vector).

Algorithm 10 Bayesian Optimization Algorithm (BOA)

1: Let $D_{1:n} = \{(\theta_i, \hat{\eta}(\theta_i), \xi_i)\}|_{i=1}^{n}$.
2: Compute the matrix of covariances $\mathbf{K}$.
3: Select the next policy to evaluate: $\theta_{n+1} = argmax_\theta E_{P(\eta(\theta)|D)}\left[I(\theta)\right]$.
4: Execute the policy θ_{n+1} for E episodes.
5: Compute Monte-Carlo estimate of expected return $\hat{\eta}(\theta_{n+1}) = \frac{1}{E}\sum_{\xi \in \xi_{n+1}} \bar{R}(\xi)$
6: Update $D_{1:n} = D_{1:n} \cup (\theta_{n+1}, \hat{\eta}(\theta_{n+1}))$
7: Return to step 2.

5.2 Bayesian Optimization for RL

Algorithm 10 outlines the basic loop of the BO routine discussed above. Line 1 assumes a batch of data of the form, $D_{1:n} = \{(\theta_i, \hat{\eta}(\theta_i), \xi_i)\}|_{i=1}^{n}$. We use $\hat{\eta}$ to indicate that we only observe a Monte-Carlo estimate of the expected return. The Monte-Carlo estimate for policy θ_i is computed using a set of trajectories ξ_i. Given data $D_{1:n}$, the surface of the expected return is modeled using the GP. The kernel matrix $\mathbf{K}$ is pre-computed in line 2 for reuse during maximization of the EI. Line 3 maximizes the EI. For this purpose, any appropriate optimization package can be used. To compute the expected improvement the GP posterior distribution $P(\eta(\theta_{n+1}|D_{1:n}))$ must be computed. This entails computing the mean function $m(\theta_{n+1})$, the vector of covariances $\mathbf{k}(\theta_{n+1}, \theta)$, and performing the required multiplications. Additional computational costs introduced into the mean and kernel functions will impact the computational cost of the optimization. Our modifications to the underlying model will increase these costs, but lead to more efficient search of the objective function space. The additional costs must be balanced against the cost of evaluating the objective function. Once selection is completed, new trajectories are generated from the selected policy (line 4), and an estimate of the expected return is recorded (lines 5 and 6).

5.2.1 Model-Free RL via Bayesian Optimization: A Behavior-based Kernel

The kernels used in prior work were not carefully designed to take advantage of the special properties of RL problems. Our goal is to design a kernel for the RL setting that leverages trajectory data to compute a domain independent measure of policy relatedness. Consider BO algorithms as a kind of space filling algorithm. Wherever sufficient uncertainty exists, the algorithm will aim to select a point to fill that space thereby reducing uncertainty in the vicinity of the selected point. The kernel function defines the volume to be filled. It is important for the development of BO algorithms for RL that kernel functions are robust to the parametrization of the policy space. Most kernel functions do not have this property. For instance, squared exponential kernels require that policies have a finite and fixed number of parameters. The kernel cannot compare non-parametric policies. In this case, individual policies can have distinct structures preventing their comparison using this form of kernel function. We seek a kernel function which is useful for comparing policies with distinct structural forms.

To construct an appropriate volume of uncertainty for the RL problem we propose relating policies by their behaviors. We define the behavior of a policy to be the associated trajectory density $P(\xi|\theta)$. Below, we discuss how to use the definition to construct a kernel function and how to estimate the kernel values without learning transition and reward functions.

First, we show how this density relates to policy returns by proving the following theorem:

Theorem 1. *For any θ_i, and θ_j,*

$$|\eta(\theta_i) - \eta(\theta_j)| \leq \frac{Rmax}{\sqrt{2}} \left[\sqrt{KL(P(\xi|\theta_i)||P(\xi|\theta_j))} + \sqrt{KL(P(\xi|\theta_j)||P(\xi|\theta_i))}\right].$$

Proof: Below we establish the upper bound stated above. To begin we rewrite the absolute value of the difference in expected returns,

$$|\eta(\theta_i) - \eta(\theta_j)| = \left| \int R(\xi) P(\xi|\theta_i) d\xi - \int R(\xi) P(\xi|\theta_j)\, d\xi \right| = \left| \int R(\xi)(P(\xi|\theta_i) - P(\xi|\theta_j))\, d\xi \right|.$$

By moving the absolute value into the integrand we upper bound the difference,

$$\left| \int R(\xi)(P(\xi|\theta_i) - P(\xi|\theta_j))\, d\xi \right| \leq \int |R(\xi)(P(\xi|\theta_i) - P(\xi|\theta_j))|\, d\xi.$$

We define a new quantity $Rmax$ bounding the trajectory reward from above. The trajectory reward is simply the sum of rewards at each trajectory step. Rmax is defined to be $T \cdot Rmax$, where T is the maximal trajectory length, and $Rmax$ is the maximal immediate reward. Using the $Rmax$ quantity we construct a new bound,

$$\int |R(\xi)(P(\xi|\theta_i) - P(\xi|\theta_j))|\, d\xi \leq Rmax \int |P(\xi|\theta_i) - P(\xi|\theta_j)|\, d\xi.$$

Expressing the difference in returns as the product of a constant and a term depending only on the variational difference in the trajectory densities. An upper bound for the variational distance was developed by (Pinsker, 1964) (the inequality was recently generalized in (Reid & Williamson, 2009)). The inequality states that $\frac{1}{2}(V(P,Q))^2 \leq KL(P||Q)$ where V is the variational distance, $\int |(P(x) - Q(x))| dx$, and $KL(P,Q)$ is the Kullback Leibler divergence,

$$KL(P(\xi|\theta_i)||P(\xi|\theta_j)) = \int P(\xi|\theta_i) log \left(\frac{P(\xi|\theta_i)}{P(\xi|\theta_j)} \right) d\xi.$$

We can use Pinsker's inequality to upper bound the variational distance,

$$Rmax \int |P(\xi|\theta_i) - P(\xi|\theta_j)|\, d\xi \leq Rmax\, \sqrt{2}\, \sqrt{KL(P(\xi|\theta_i)||P(\xi|\theta_j))}.$$

Finally, we use the fact that the variational distance is symmetric $\int |(P(x) - Q(x))|dx = \int |(Q(x) - P(x))|dx$ to argue that,

$$|\eta(\theta_i) - \eta(\theta_j)| \leq \frac{Rmax}{\sqrt{2}} \left[\sqrt{KL(P(\xi|\theta_i)||P(\xi|\theta_j))} + \sqrt{KL(P(\xi|\theta_j)||P(\xi|\theta_i))}\right].$$

Hence, this simple bound relates the difference in returns of two policies to the trajectory density $\square$.

Importantly, the bound is a symmetric positive measure of the distance between policies. It bounds, from above, the absolute difference in expected value, and reaches zero only when the divergence is zero (the policies are the same). Additionally, computing the tighter variational bound, $Rmax \int |P(\xi|\theta_i) - P(\xi|\theta_j)|\, d\xi$, inherently requires knowledge of the domain transition models. Alternatively, the log term of the KL-divergence is a ratio of path probabilities. Given a sample of trajectories the ratio can be computed with no knowledge of the domain model. This characteristic is important when learned transition and reward functions are not available. Our goal is to incorporate the final measure of policy relatedness into the surrogate representation of the expected return. Unfortunately, the divergence function does not meet the standard requirements for a kernel ((Moreno *et al.*, 2004)). It is not a positive semidefinite function (Rasmussen & Williams, 2005). To transform the bound into a valid kernel we first define a function,

$$D(\theta_i, \theta_j) = \sqrt{KL(P(\xi|\theta_i)||P(\xi|\theta_j))} + \sqrt{KL(P(\xi|\theta_j)||P(\xi|\theta_i))},$$

and define the covariance function to be the negative exponential of D,

$$K(\theta_i, \theta_j) = exp(-\alpha \cdot D(\theta_i, \theta_j)).$$

The kernel has a single scalar parameter α controlling its width. This is precisely what we sought, a measure of policy similarity which depends on the action selection decisions. The kernel compares behaviors rather than parameters, making the measure robust to changes in policy parametrization.

5.2.2 Estimation of the Kernel Function Values

Below we discuss using estimates of the divergence values in place of the exact values for $D(\theta_i, \theta_j)$. Computing the exact KL-divergence requires access to a model of the decision process and is a computationally demanding process. No closed form solution is available.

The divergence must be estimated. In this work we elect to use a simple Monte-Carlo estimate of the divergence. The divergence between policy θ_i and θ_j is approximated by,

$$\hat{D}(\theta_i, \theta_j) = \sum_{\xi \in \xi_i} log\left(\frac{P(\xi|\theta_i)}{P(\xi|\theta_j)}\right) + \sum_{\xi \in \xi_j} log\left(\frac{P(\xi|\theta_j)}{P(\xi|\theta_i)}\right),$$

using a sparse sample of trajectories generated by each policy respectively (ξ_i represents the set of trajectories generated by policy θ_i). Because of the definition of the trajectory

density, the term within the logarithm reduces to a ratio of action selection probabilities,

$$log\left(\frac{P(\xi|\theta_i)}{P(\xi|\theta_j)}\right) = log\left(\frac{P_0(s_0)\prod_{t=1}^{T} P(s_t|s_{t-1},a_{t-1})P_\pi(a_{t-1}|\phi(s_{t-1}),\theta_i)}{P_0(s_0)\prod_{t=1}^{T} P(s_t|s_{t-1},a_{t-1})P_\pi(a_{t-1}|\phi(s_{t-1}),\theta_j)}\right)$$
$$= \sum_{t=1}^{T} log\left(\frac{P_\pi(a_t|s_t,\theta_i)}{P_\pi(a_t|s_t,\theta_j)}\right),$$

easily computed without a model.

A problem arises when computing the Expected Improvement (Line 3 of the BOA). Computing the conditional mean and covariance for new points requires the evaluation of the kernel for policies which have no trajectories present in the data set. We elect to use an importance sampled estimate of the divergence, because we do not have access to learned transition and reward functions,

$$\hat{D}(\theta_{new},\theta_j) \quad = \quad \sum_{\xi\in\xi_j} \frac{P(\xi|\theta_{new})}{P(\xi|\theta_j)} log\left(\frac{P(\xi|\theta_{new})}{P(\xi|\theta_j)}\right) + log\left(\frac{P(\xi|\theta_j)}{P(\xi|\theta_{new})}\right).$$

Though the variance of this estimate can be large our empirical results show that errors in the divergence estimates, including the importance sampled estimates, do not negatively impact performance. Alternative methods of estimating f-divergences (KL-divergence being a specific case) have been proposed in the literature (Nguyen *et al.*, 2007), and can be used for future implementations.

5.2.3 Model-Based RL via Bayesian Optimization

In this section we discuss our approach to incorporating domain models into the BO algorithm. The behavior based kernel has some important limitations. First, the kernel can only compare stochastic policies. The KL divergence is only defined for policies with overlapping support. Second, the upper bound used to construct the kernel function is

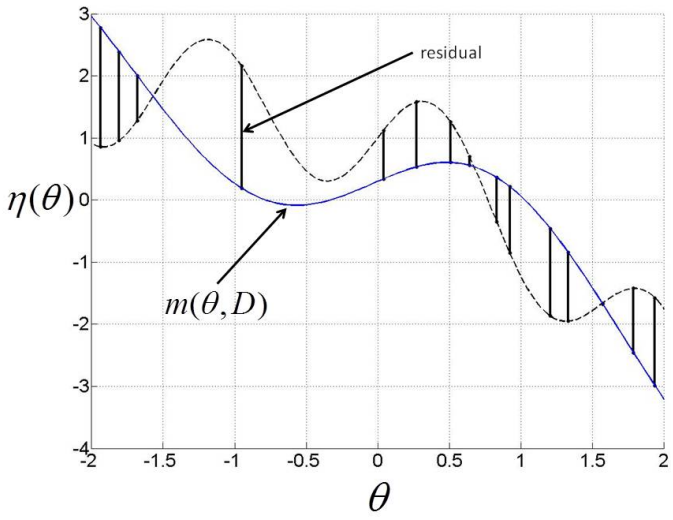
currently loose. This can lead to excessive exploration when the kernel exaggerates the differences between policies. To overcome these problems we look to make direct predictions of future states by learning transition and reward functions from the trajectory data.

If the agent has access to an accurate simulator then optimization of the simulated surface will yield optimal policies. Unfortunately, we cannot rely on the availability of an accurate simulator. Moreover, if the simulator has costs comparable to evaluating the policy in the MDP then we gain nothing by incorporating it into the BO algorithm. Presumably an accurate simulator will take as long to run as the actual system; perhaps longer if it is a physical simulator. The simulator must reduce the costs of generating trajectories. Naturally, this constraint will introduce errors in the simulation. To overcome problems stemming from the predictive errors we propose combining the output of the simulator with the output of a residual model.

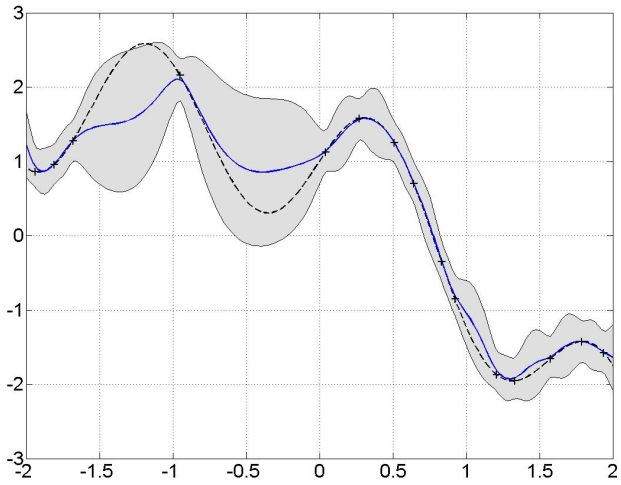
Our Model-Based Bayesian Optimization Algorithm (MBOA) builds an approximate simulator from the observed trajectory data. The simulator is used to generate roll-out estimates of policy performance. A roll-out is performed by initializing the state of the approximate simulator and then executing the desired policy until the simulator terminates. Simulated trajectories are generated from the approximate trajectory density,

$$\hat{P}(\xi|\theta) = P_0(s_0) \prod_{t=1}^{T} \hat{P}(s_t|s_{t-1}, a_{t-1}) P_\pi(a_{t-1}|\phi(s_{t-1}), \theta).$$

We write $\hat{P}$ to indicate that the transition function was learned from the observed trajectory data in $D_{1:n}$. From a fixed sample of N simulated trajectories we compute the



(a)



(b)

Figure 5.3: (a) The relationship between the surface of $m(\theta, D)$ and the objective function. Both surfaces are observed at a fixed set of points. The values of the surfaces at these points compose the vectors $\mathbf{y}$ and $\mathbf{m}(D_{1:n}, \theta)$ respectively. The magnitude of the residuals are depicted as vertical bars. We build a GP model of these residual values. (b) The complete model combines the function $m(D_{1:n}, \theta)$ (as depicted above) with the GP model of the residuals. The solid blue line corresponds to the corrected mean of the model. The shaded area depicts two standard deviations from the mean.

approximate expected return,

$$m(D_{1:n}, \theta_i) = \hat{\eta}(\theta_i) = \frac{1}{N} \sum_{j=1}^{N} \hat{R}(\xi_j). \tag{5.2}$$

The function $\hat{R}$ indicates the reward function learned from data. The function $m(D_{1:n}, \theta_i)$ is the roll-out estimate of the expected return for policy θ_i; $m(D_{1:n}, .)$ represents an approximation of the objective function surface. The relationship between the function $m(D_{1:n}, \theta_i)$ and the objective function are illustrated in Figure 5.4(a).

To make use of this Monte-Carlo estimate we replace the zero mean function of the GP prior distribution with $m(D_{1:n}, \theta_i)$. The predictive distribution of the GP changes to be Gaussian with mean,

$$\mu(\theta_{n+1}|D_{1:n}) = m(D_{1:n}, \theta_{n+1}) + \mathbf{k}(\theta_{n+1}, \theta)\mathbf{K}(\theta, \theta)^{-1}(\mathbf{y} - \mathbf{m}(D_{1:n}, \theta)),$$

where $\mathbf{m}(D_{1:n}, \theta)$ is the column vector of Monte-Carlo estimates for each policy in D; $\mathbf{m}(D_{1:n}, \theta_i) = m(D_{1:n}, \theta_i)$ (This vector must be recomputed whenever new trajectories are added to the data). The new predictive mean is a sum of the simulated expected return and the GP's prediction of the residual. The model of the residuals directly compensates for errors introduced by the simulator (Figure 5.4(c)). The variance is unchanged. The role of the GP has changed from directly modeling the surface of the expected return to modeling the disagreement between the approximate simulator and the observed returns.

The danger of allowing transition and reward model bias presents itself during the policy selection step of the BO algorithm. Consider, the following degenerate case. The approximate simulator returns pessimistic approximations of the expected return for all

policies. If the estimated values are sufficiently below the currently observed maximum, η_{max}, the MEI will be zero, and exploration will cease.

To prevent early termination of exploration, we assume a new model of the expected return,

$$\eta(\theta) = (1 - \beta)f(\theta) + \beta g(\theta).$$

The new model is a tradeoff between two functions. It is governed by the parameter β. We model the function $f(\theta)$ with a zero mean GP, and we model function $g(\theta)$ with a GP distribution that uses the model-based mean $m(D_{1:n}, \theta)$ introduced above. The kernel of both GP priors is identical. The resulting distribution for $\eta(\theta)$ is a GP with mean $\beta m(D, \theta)$ and covariance computed using the kernel function. This change impacts the predicted mean which now weights the model estimate by β,

$$\mu(\theta_{n+1}|D_{1:n}) = \beta m(D_{1:n}, \theta_{n+1}) + k(\theta_{n+1}, \theta)K(\theta, \theta)^{-1}(\eta(\theta) - \beta m(D_{1:n}, \theta)).$$

The variance changes to incorporate a factor $c(\beta)$.

We choose to optimize the β parameter using maximum likelihood. An alternative strategy, familiar to GP practitioners, is to specify a prior distribution encoding knowledge of the β parameter. As the system accumulates new data the disagreement between the vector of observed returns and the mean function can be computed. The magnitude of the residuals will impact the optimized β value. To adjust the parameter β based on the agreement between the estimates and the observed data we maximize the log likelihood of the GP prior. For a GP with mean βm and covariance matrix $\mathbf{K}$ the log likelihood of the data is,

$$P(\eta(\theta)|D_{1:n}) = -\frac{1}{2}(\eta(\theta) - \beta m)^t K^{-1}(\eta(\theta) - \beta m) - \frac{1}{2}log|K + \sigma^2 I| - \frac{n}{2}log(2\pi).$$

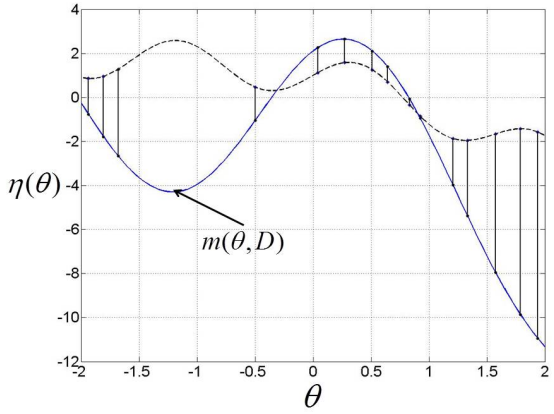
Taking the gradient of the log likelihood and setting the equation to zero we get a simple expression for β,

$$\beta = \frac{\eta(\theta)^t K^{-1} m}{m^t K^{-1} m}.$$

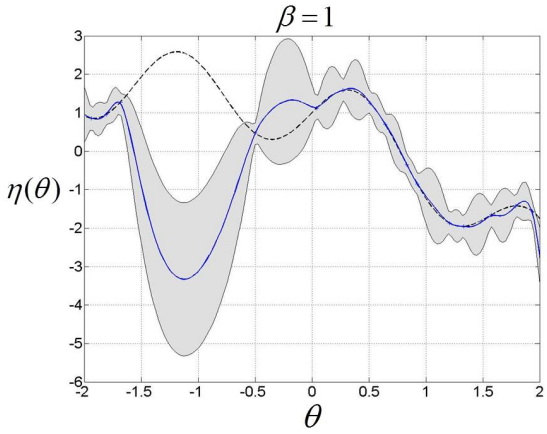
Effectively, optimizing the value of β, prior to maximizing the expected improvement, allows the model to control the impact of the simulator on the predictions. Intuitively, the algorithm can return to the performance of the unmodified BOA when errors in the simulator are large (by setting β to zero).

Algorithm 11 outlines the steps necessary to incorporate the new model into the BOA. MBOA takes advantage of trajectory data by building an approximate simulator of the expected return. Like the BBK the mean function used by MBOA only requires policies to output actions. It is oblivious to the internal structure of policies. Additionally, by contrast to the BBK, MBOA can compare deterministic policies.

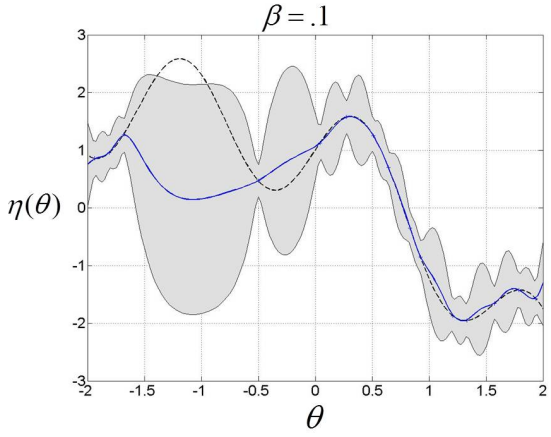
Most work in model-based RL assumes the learned transition and reward functions accurately represent the true functions. Unfortunately, this is difficult to guarantee in real-world tasks. Due to our limited knowledge of the real-world domains, transition and reward models frequently exhibit model bias. Bias can occur when few data points are available, or when the chosen class of functions cannot accurately represent the target function. MBOA aims to overcome both of these sources of bias by combining a weighted mean function with a residual model. By construction MBOA can ignore the approximate simulator if it produces systematic errors, and can benefit from the simulator when the predictions are accurate. This reduces the modeling effort on part of the designer. This is an advantage when there are restrictions on prior knowledge or on computational resources. In the results section we show empirical examples where MBOA benefits from the use of simplified transition and reward functions.



(a) An example of the degenerate case (objective function marked by dashed line).



(b) The corrected model with control parameter $\beta = 1$. After correction the model underestimates the objective.



(c) The corrected model with control parameter $\beta = .1$. Reducing the influence of the roll-out estimates improves the predictive quality of the model.

Algorithm 11 Model-based Bayesian Optimization Algorithm (MBOA)

1: Let $D_{1:n} = \{\theta_i, \eta(\theta_i), \xi\}|_{i=1}^{n}$.
2: Run the $LearnModel(D_{1:n})$ function.
3: Compute the vector $m(D_{1:n}, \theta)$ using the approximate simulator.
4: Optimize β by maximizing the log likelihood.
5: Select the next point in the policy space to evaluate: $\theta_{n+1} = argmax_{\theta} E(I(\theta)|D_{1:n})$.
6: Execute the policy with parameters θ_{n+1} in the MDP.
7: Update $D_{1:n+1} = D_{1:n} \cup (\theta_{n+1}, \eta(\theta_{n+1}), \xi_{n+1})$
8: Return to step 2.

5.3 Experimental Results

We examine the performance of MBOA and BOA with the behavior based kernel (BBK) in 5 benchmark RL tasks including a mountain car task, a Cart-pole balancing task, a 3-link planar arm task, an acrobot swing up task, and a bicycle balancing task. We use the bicycle simulator originally introduced in (Randløv & Alstrøm, 1998).

Comparisons are made between MBOA, BOA with the BBK, the model-based DYNA-Q algorithm, and four model-free approaches: 1. BOA with a squared exponential kernel. 2. OLPOMDP. 3. Q-Learning with CMAC function approximation. 4. LSPI.

5.3.1 Experiment Setup

We detail the special requirements necessary to implement each algorithm in this section.

For all experiments, except Cart-pole, MBOA, BOA, and OLPOMDP algorithms search for parametric soft-max action selection policies,

$$P(a|s) = \frac{exp(\theta_a \cdot \phi(s))}{\sum_{y \in A} exp(\theta_y \cdot \phi(s))}.$$

The parameters θ_y are the set of policy parameters associated with action y. The function

$\phi(s)$ computes features of the state. In the Cart-pole experiments, a linear policy maps directly to the action a,

$$a = \theta \cdot \phi(s) + \epsilon.$$

Epsilon is a small noise parameter used in algorithms requiring stochastic policies.

MBOA and BOA. To use the GP prior in MBOA and BOA we must specify a kernel and mean function. The squared exponential kernel,

$$k(\theta_i, \theta_j) = exp(-\frac{1}{2}\rho(\theta_i - \theta_j)^T(\theta_i - \theta_j)), \tag{5.3}$$

with a single scaling parameter ρ, was used in all experiments. More complex kernels can be selected and optimized. However, we were able to get positive results with this simple kernel for our experiments. The ρ parameter was tuned for each experiment, and the same value was used in both MBOA and BOA. The mean function of BOA is the zero function, and MBOA employs the model-based mean function discussed above.

It is necessary to optimize the expected improvement for all of the BO algorithms. For this purpose we make use of two simple gradient-free black box optimization algorithms. The DIRECT algorithm detailed in (Jones *et al.*, 1993) is used for all tasks except bicycle. DIRECT is poorly suited for problems with more than 15 dimensions. In the bicycle riding domain the policy has 100 dimensions. In this case we use the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm detailed in (Hansen, 2006). Both DIRECT and CMA-ES require specifying upper and lower bounds on the policy parameters. We specify an upper bound of 1 and a lower bound of -1 for each dimension of the policy. These bounds hold for all experiments reported below.

To implement the MBOA algorithm the designer must define a class of domain models. For the experiments reported below linear models were used. The models are of the

form,

$$s_i^t = w_i \cdot \phi(s^{t-1}, a^{t-1})', r^t = w_r \cdot \phi(s^{t-1}, a^{t-1}, s^t)', \tag{5.4}$$

where s_i^t is the i^{th} state variable at time t, w_i is the weight vector for the i^{th} state variable, and $\phi(s^{t-1}, a^{t-1}, s^t)'$ is a column vector of features computed from the states and actions (and next states in the case of the reward model). Parameters are estimated from data using standard linear regression.

DYNA-Q. We make two slight modifications to the DYNA-Q algorithm. First, we provide the algorithm with the same linear models employed by MBOA. These are models of continuous transition functions which DYNA-Q is not normally suited to handle. The problem arises during the sampling of previously visited states during internal reasoning. To perform this sampling we maintain a dictionary of past observations and sample visited states from it. These continuous states are discretized using a CMAC function approximator. The second change we make is to disallow internal reasoning until a trial is completed. To reduce the computational cost reasoning between steps is not allowed. After each trial DYNA-Q is allowed 200000 internal samples to update its Q-function. This was to insure that during policy selection DYNA-Q was allowed computational resources comparable to the resources used by MBOA and BOA with the BBK.

Q-Learning with CMAC function approximation ϵ-Greedy exploration is used in the experiments (ϵ is annealed after each step). The discretization of the CMAC approximator was chosen to give robust convergence to good solutions.

OLPOMDP OLPOMDP is a simple gradient based policy search algorithm ((Baxter *et al.*, 2001)). The OLPOMDP implementations use the same policy space optimized by the BO algorithms.

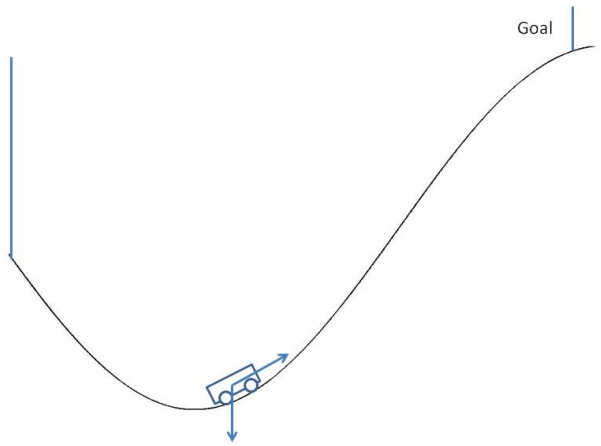
LSPI. LSPI results are reported in the cart-pole, acrobot, and bicycle tasks. We

do not report mountain car or arm results because no tested combinations of basis functions and exploration strategies yielded good performance. We attempted radial basis, polynomial basis, and hybrid basis. Our exploration strategy uses policies returned from the LSPI optimization with added noise (including fully random exploration). For those domains with results radial basis functions were used with centers located to match the CMAC function approximator.

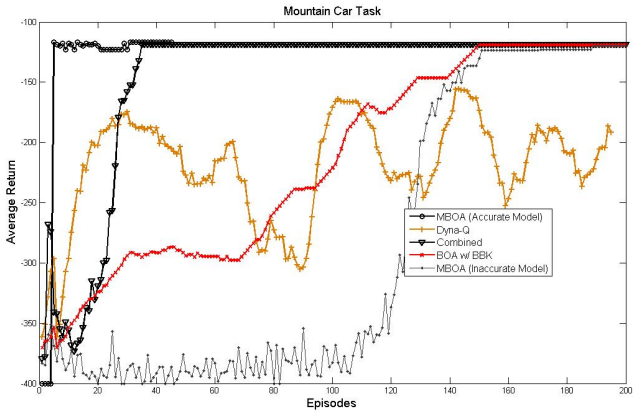
5.3.2 Mountain Car Task

In the mountain car domain, shown in Figure 5.4(d), the goal is to accelerate a car from a fixed position at the base of a hill to the hill's apex. Our implementation of the mountain car task uses eight features derived from the standard state variables (velocity and location of the car). The control policy selects from two actions (applying acceleration forward or to the rear). The policy has sixteen dimensions. The reward function penalizes the agent -1 for each step taken to reach the goal. Agents are allowed a maximum of 400 actions for each episode.

The sparse reward signal makes Mountain Car an ideal experiment for illustrating the importance of directed exploration based on differences in policy behavior. Approaches based on random exploration and exploration weighted by returns are poorly suited to this kind of domain (we have excluded the other model-free methods from the graph because they fail to improve within 200 episodes). Visual inspection of the performance of BOA shows that many of the selected policies, which are unrelated according to the squared exponential kernel, produce similar action sequences when started from the initial state. This redundant search is completely avoided when generalization is controlled by the BBK.



(d)



(e)

Figure 5.4: (a) Mountain Car illustration. (b) Mountain Car Task: We report the total return per episode averaged over 40 runs of each algorithm.

Note that the performance of all kernel methods depends on the settings of the parameters. Optimization of the hyper parameters is not the focus of this work. However, the parameters of the BBK (and any other kernel used with MBOA) can be automatically optimized using any standard method of model selection for GP models ((Rasmussen & Williams, 2005)). We elect to simply set the parameter α to 1 for all remaining experiments.

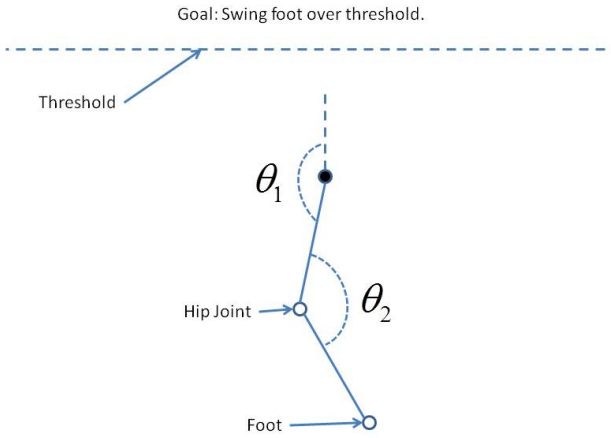
When accurate domain models can be learned, the Monte-Carlo estimates of the expected return computed by the MBOA algorithm will accurately reflect the true objective. Therefore, if the domain models can be accurately estimated with few data points MBOA will rapidly identify the optimal policy. To illustrate this we hand-constructed a set of features for the linear domain models used in the mountain car task. Depicted in Figure 5.4(e) are the results for MBOA with high quality hand-constructed features. Once the domain models are estimated, only four episodes are needed to yield accurate domain models. Once accurate domain models are available MBOA immediately finds an optimal policy. Typically, the insight used to construct model features is not available in complex domains. To understand the performance of MBOA in settings with poor insight into the correct domain model class, we remove the hand constructed features from the linear models and examine the resulting performance. With poor models, the performance of the MBOA algorithm degrades. As shown in the figure, the performance of MBOA is no longer better than the performance of standard BOA with the model-free BBK. Because of its special properties the BBK can be usefully combined with the model-based mean function used in MBOA. We show the performance of this combination, using the BBK as the kernel function for the MBOA algorithm, labeled Combined, in Figure 5.4(e). The results are comparable to the case where MBOA has access to high quality domain models.

The performance of MBOA should be contrasted with the performance of DYNA-Q. During DYNA-Q's internal simulations errors in the estimated domain models negatively impact the accuracy of the Q-value estimates. This leads to poor performance in the actual task. In contrast, MBOA is specifically constructed to mitigate the impact of inaccurate domain models. Eventually, even if the domain models have significant errors MBOA can still identify the optimal policy. No such guarantee can be made by DYNA-Q (or most model-based RL algorithms).

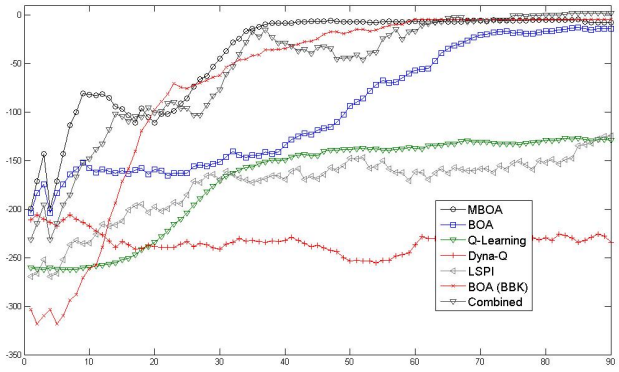
5.3.3 Acrobot Task

In the acrobot domain, depicted in Figure 5.5(a), the goal is to swing the foot of the acrobot over a specified threshold by applying torque at the hip. Details of the implementation can be found in (Sutton & Barto, 1998). Four features are used for the soft-max policy resulting in twelve policy parameters.

We were unable to construct accurate linear domain models for this task. Consequently, the DYNA-Q algorithm is unable to find a good policy. MBOA is able to compensate for the domain model errors and exhibits the best performance (some policies are accurately simulated by the poor models). The performance of the BBK is less pronounced in this domain. The policy class generates a varied set of behaviors such that small changes in the policy parameters lead to very different action sequences. Therefore, more behaviors must be searched before good policies are identified. Even so, the behavior based kernel does outperform BOA with squared exponential kernel and it is competitive with the performance of MBOA. In contrast to the mountain car task, combining the BBK with MBOA does not yield improved performance. The performance is comparable to MBOA.



(a)



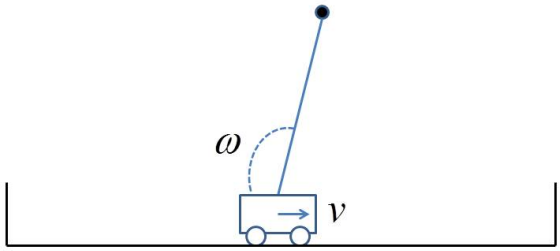
(b)

Figure 5.5: (a) Acrobot illustration. (b) We report the total return per episode averaged over 40 runs of each algorithm.

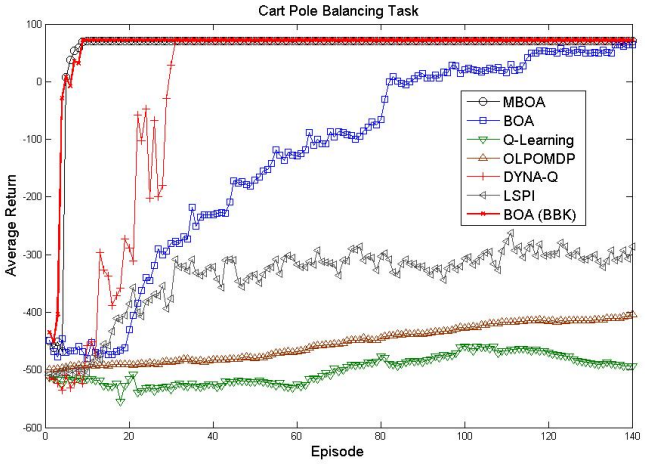
5.3.4 Cart-pole Task

In the Cart-pole domain, depicted in Figure 5.6(a), the agent attempts to balance a pole fixed to a movable cart. The policy selects the magnitude of the change in velocity (positive or negative) applied to the base of the cart. The policy is a function of four features (v, v', ω, ω'), the velocity, change in velocity, angle of the pole, and angular velocity of the pole. The reward function gives a positive reward for each successful step. A penalty is added to this reward when pole angles deviate from vertical, and when the location of the cart deviates from the center. A large positive reward is received after 1000 steps if the pole has been kept upright and the cart has remained within a fixed boundary. Episodes terminate early if the pole falls or the cart leaves its specified boundary.

Figure 5.6 shows the results for the Cart Pole task. Clearly, the BBK outperforms all of the model-free competitors including BOA with the squared exponential kernel. Its performance is comparable to MBOA despite being fully model-free. In this task, the linear domain models are highly accurate. MBOA effectively employs the models to rapidly identify the optimal policy. By comparison, all other methods, except BOA(BBK), must accumulate more data to identify a good policy. BOA consistently identifies a policy which balances the pole for the full 1000 steps. Q-Learning and OLPOMDP require at least 1250 episodes before similar performance is achieved. LSPI converges faster but cannot match the performance of either the BOA or MBOA algorithm. Please note that our problem is distinct from the original LSPI balancing task in the following ways: 1. The force applied by the agent is more restricted. 2. The cart is constrained to move within a fixed boundary. 3. The reward function penalizes unstable policies. After 30 episodes the DYNA-Q algorithm has found a solution comparable to that of MBOA.



(a)



(b)

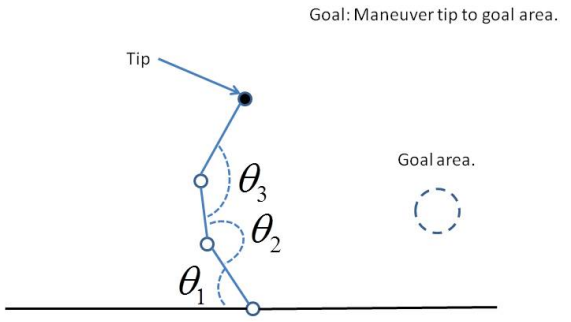
Figure 5.6: (a) Cart-pole illustration. (b) Cart Pole Task: We report the total return per episode averaged over multiple runs of each algorithm.

This performance is unsurprising given the accuracy of the estimated domain models.

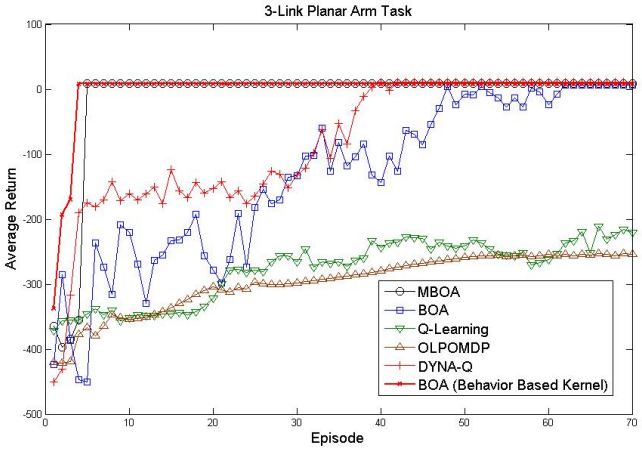
5.3.5 3-Link Planar Arm Domain

In the 3-link Planar Arm task the goal is to direct the arm tip to a target region on a 2 dimensional plane. The state of the Planar Arm is depicted in Figure 5.7(a). Each of three controllers independently outputs a torque (-1,1) for one arm joint. Each controller is a soft-max action selection policy. Only two features are needed for each controller resulting in 12 total policy parameters (policy parameters are jointly optimized). The features are composed of x and y displacements from the center of the target location to the arm tip location. The reward function penalizes the agent proportional to the distance between the arm tip and the target location. A positive reward is received when the arm tip is placed within the goal space.

The performance of each algorithm is shown in Figure 5.7(b). The generalization of the divergence-based kernel, BOA(BBK), is particularly powerful in this case. Much of the policy space is quickly identified to be redundant and is excluded from the search. The agent fixes on an optimal policy almost immediately. Like the Cart-pole task, the transition and reward function of the arm task are linear functions. Once the functions are estimated MBOA quickly identifies the optimal policy (the mean function accurately reflects the true surface). Additional data is needed before the BOA algorithm identifies policies with similar quality. The other model-free alternatives require 1000 episodes before converging to the same result. DYNA-Q found a policy comparable to the MBOA algorithm, but required more experience. In the three tasks discussed above the MBOA algorithm makes better use of the available computational resources and remains robust to model bias.



(a)



(b)

Figure 5.7: (a) Planar arm illustration. (b) We report the total return per episode averaged over multiple runs of each algorithm.

5.3.6 Bicycle Balancing

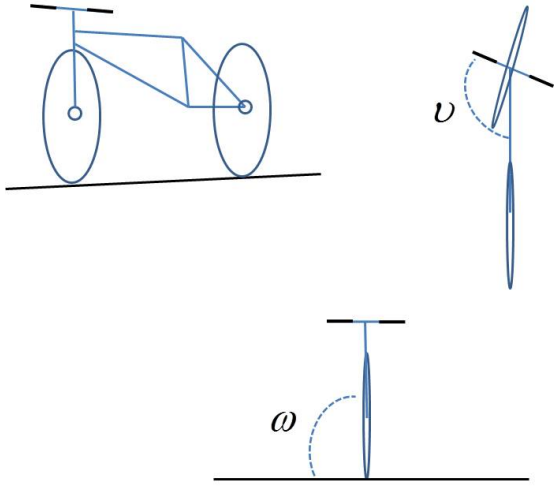
Agents in the bicycle balancing task the agent must keep the bicycle balanced for ten minutes of simulated time. For our experiments we use the simulator originally introduced in (Randløv & Alstrøm, 1998). The state of the bicycle, depicted in Figure 5.8(a), is defined by four variables $(\omega, \dot{\omega}, \nu, \dot{\nu})$. The variable ω is the angle of the bicycle with respect to vertical, and $\dot{\omega}$ is its angular velocity. The variable ν is the angle of the handlebars with respect to neutral, and $\dot{\nu}$ is the angular velocity. The goal of the agent is to keep the bicycle from falling. Falling occurs when $|\omega| > \pi/15$. The same discrete action set used in (Lagoudakis *et al.*, 2003) is used in our implementations. Actions have two components. The first component is the torque applied to the handlebars $T \in (-1, 0, 1)$, and the second component is the displacement of the rider in the saddle $p \in (-.02, 0, .02)$. Five actions are composed from these values $(T, p) \in ((-1, 0), (1, 0), (0, -.02), (0, .02), (0, 0))$. The reward function is defined to be the squared change in ω, $(\omega_t - \omega_{t+1})^2$, at each time step. An additional -10 penalty is imposed if the bicycle falls before time expires. Rewards are discounted in time insuring that longer runs result in smaller penalties. In our implementation the set of 20 features introduced in (Lagoudakis *et al.*, 2003) were used. The softmax policy has 100 parameters. Please note that our LSPI implementation of bicycle does not benefit from the design decisions introduced in (Lagoudakis *et al.*, 2003) which included a carefully designed shaping reward and short example trajectories.

In Figure 5.8(b) we report the performance of MBOA and LSPI. All other implementations, including MBOA combined with the BBK, show no improvement in 300 episodes. In this case the BBK performs poorly. It overestimates the distance between policies with dissimilar action selection distributions. In the bicycle riding task individual sequences of oscillating actions can produce similar state trajectories. A policy exe-

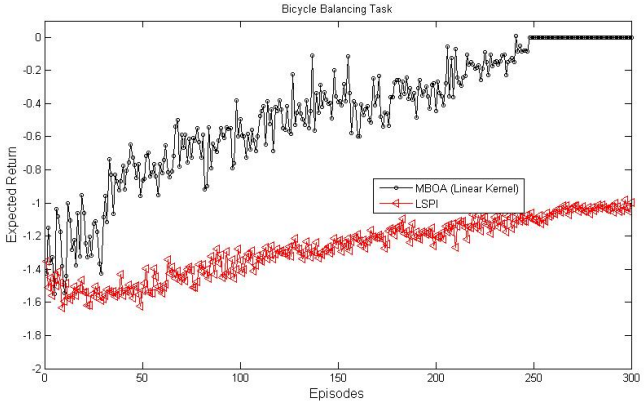
cuting right,left,right,left,.... behaves much like a policy executing left,right,left,right,... even though the action selection distributions may be arbitrarily dissimilar. This is a problem with the BBK that must be addressed in future work. By contrast, after 300 episodes MBOA has found a balancing policy. For this experiment we used a linear kernel function with automatic relevance determination. The parameters of the linear kernel function were optimized by maximum likelihood after each policy execution. Before optimization of the hyper-parameters the algorithm was made to choose 20 policies with the uninformed values. MBOA uses very simple and highly inaccurate linear models in this task. Errors in the linear models are significant enough to cause DYNA-Q to fail. MBOA exhibits robust performance despite the predictive errors caused by model bias. It converges to a near optimal policy and exhibits excellent data efficiency.

5.4 Related Work

By extending the work on BO we place our research squarely within the growing body of literature on Bayesian RL. Most closely related is the extensive work by (Lizotte, 2008a) adapting BO methods to the RL problem, which we extend by incorporating approximate domain models and by defining a behavior based kernel function. Other related work includes ((Engel *et al.*, 2005)) which modeled the value function as a Gaussian Process. (Engel *et al.*, 2005) focused primarily on the problem of estimation, did not consider parameterized policies, and did not explore the use of domain models for improving data efficiency. Numerous model-based Bayesian approaches, including (Dearden *et al.*, 1999; Strens, 2000; Duff, 2003), take advantage of uncertainty in domain models to tackle the exploration-exploitation trade off. Like our algorithm, these methods actively explore to reduce uncertainty. However, each of these methods requires the models to be accurate



(a)



(b)

Figure 5.8: (a) Bicycle illustration. (b) Bicycle Balancing Task: We report the total return per episode averaged over multiple runs of each algorithm (15 runs of MBOA and 30 runs of LSPI).

to insure eventual convergence. We do not have this stringent requirement.

Work by (Kakade, 2001) presented a metric based on the Fisher information to derive the natural policy gradient update. (Kakade, 2001) was able to show natural policy gradient methods outperforming standard gradient methods in a difficult Tetris domain. Follow up work by (Bagnell & Schneider, 2003) proposed pursuing a related idea within the path integral framework for RL (the same framework of this paper). Their work considered metrics defined as functions on the distribution over trajectories generated by a fixed policy $P(\xi|\theta)$. In contrast to our goals both works focus on iteratively improving a policy via gradient descent. Furthermore, no explicit attention is paid to using the metric information to guide the exploratory process. However, the insight that policy relationships should be expressed as functions of the trajectory density has played a key role in the development of our behavior based kernel.

Related to our proposed kernel function is recent work in (Peters *et al.*, 2010) and (Kober & Peters, 2010). (Peters *et al.*, 2010) used a divergence-based bound to control exploration. Specifically, they attempt to maximize the expected reward subject to a bound proportional to the KL-divergence between the empirically observed state-action distribution and the state-action distribution of the new policy. The search for a new policy is necessarily local, restricted by the bound to be close to the current policy. By contrast, our work uses the divergence as a measure of similarity, allowing for aggressive search of the policy space. Related work by (Kober & Peters, 2010) derives a lower bound on the importance sampled estimate of the expected return, as was done in (Dayan & Hinton, 1997), and observes the relationship to the KL-divergence of the reward weighted behavior policy and the target policy. They derive from this relationship an EM-based update for the policy parameters. An explicit effort is made to construct the update such that exploration is accounted for. However, their method of state-dependent exploration

is still based on random perturbations of the action selection policy. Our method of exploration is instead determined by the posterior uncertainty and does not depend on the behavior policy. This is an advantage when working with real physical systems where random perturbations can damage expensive equipment.

5.5 Conclusions

General black box Bayesian Optimization algorithms have been adapted to the RL policy search problem. These methods have already shown promise in difficult domains including the learning of gaits for AIBO robots. However, previous work adapting these approaches to RL have ignored the central character of the RL problem: the sequential nature of the decision process. This paper presents two complementary approaches taking advantage of the sequential data observed by RL agents.

For model-free RL, we present a new kernel function for GP models of the expected return. The kernel compares sequences of action selection decisions from pairs of policies. We motivate our kernel by examining a simple upper bound on the absolute difference of expected returns for two arbitrary policies. We use this upper bound as the basis for our kernel function, argue that the properties of the bound insure a more reasonable measure of policy relatedness, and demonstrate empirically that this kernel can substantially speed up exploration. The empirical results show that the derived kernel is competitive with the model-based RL algorithms MBOA and Dyna-Q, and converges more quickly than the model-free algorithms tested. Additionally, we showed that in certain circumstances BBK can be combined with the MBOA algorithm to improve the quality of exploration.

We present the MBOA for model-based RL. MBOA improves on the standard BOA

algorithm by using collections of trajectories to learn an approximate simulator of the decision process. The simulator is used to define an approximation of the underlying objective function. Potentially, such a function provides useful information about the performance of unseen policies which are distant (according to a measure of similarity) from previous data points. Empirically, we show that the MBOA algorithm has exceptional data efficiency. Its performance exceeds LSPI, OLPOMDP, Q-Learning with CMAC function approximation, BOA, and DYNA-Q in all of our tasks. By correcting the simulator with a GP model of the residuals, the MBOA algorithm performs well even when the learned domain models, have serious errors. Overall, MBOA appears to be a useful step toward combining model-based methods with Bayesian Optimization for the purposes of handling approximate models and improving data efficiency.

The penalty for taking this approach is straightforward. A substantial increase in computational cost is incurred during optimization of the surrogate function. This occurs when evaluating MBOA's approximate simulator, or when estimating the values of the behavior based kernel. The increase in computational requirement is linear in the length of the trajectories. However, the cost is compounded during optimization and will make these approaches infeasible when trajectory lengths become huge. Overcoming this barrier is an important step towards taking advantage of trajectory data generated in complex domains.

Chapter 6: Bayesian Active RL

Directly specifying desired behaviors for automated agents is a difficult and time consuming process. Successful implementation requires expert knowledge of the target system and a means of communicating control knowledge to the agent. One way the expert can communicate the desired behavior is to directly demonstrate it and have the agent learn from the demonstrations, e.g. via imitation learning (as per BPS in Chapter 4) (Schaal, 1996; Argall *et al.*, 2009; Price & Boutilier, 2003) or inverse reinforcement learning (Ng & Russell, 2000). However, in some cases, like the control of complex robots or simulated agents, it is difficult to generate demonstrations of the desired behaviors. In these cases an expert may still recognize when an agent's behavior matches a desired behavior, or is close to it, even if it is difficult to directly demonstrate it. In such cases an expert may also be able to evaluate the relative qualities to the desired behavior of a pair of example trajectories and express a preference for one or the other.

Given this motivation, in this chapter we study the problem of learning expert policies via trajectory preference queries to an expert. A *trajectory preference query (TPQ)* is a pair of short state trajectories originating from a common state. Given a TPQ the expert is asked to indicate which trajectory is most similar to the target behavior. The goal of our learner is to infer the target trajectory using as few TPQs as possible. Our first contribution (Section 6.2) is to introduce a Bayesian model of the querying process along with an inference approach for sampling policies from the posterior given a set of TPQs and their expert responses. Our second contribution (Section 6.3) is to describe two active query strategies that attempt to leverage the model in order to minimize the

number of queries required. Finally, our third contribution (Section 6.4) is to empirically demonstrate the effectiveness of the model and querying strategies on four benchmark problems.

We are not the first to examine preference learning for sequential decision making. In the work of Cheng et. al. (Cheng *et al.*, 2011) action preferences were introduced into the classification based policy iteration framework. In this framework preferences explicitly rank state-action pairs according to their relative payoffs. There is no explicit interaction between the agent and domain expert. Further the approach also relies on knowledge of the reward function, while our work derives all information about the target policy by actively querying an expert. In more closely related to our work, Akraur et. al. (Akrour *et al.*, 2011) consider the problem of learning a policy from expert queries. Similar to our proposal this work suggests presenting trajectory data to an informed expert. However, their queries require the expert to express preferences over approximate state visitation densities and to possess knowledge of the expected performance of demonstrated policies. Necessarily the trajectories must be long enough to adequately approximate the visitation density. We remove this requirement and only require short demonstrations; our expert assesses trajectory snippets not whole solutions. We believe this is valuable because pairs of short demonstrations are an intuitive and manageable object for experts to assess.

6.1 Preliminaries

We explore policy learning from expert preferences in the framework of Markov Decision Processes (MDP) presented in Chapter 2. Two important points must be made for this chapter. First, the agent will not be able to observe rewards. All information about the quality of policies is garnered from interaction with the domain expert through

the querying procedure. Second, in this chapter agents select actions using stochastic parameterized action selection policies $P_\pi(a|s,\theta)$ with parameters θ. In what follows we will denote these stochastic action selection policies π_θ. In the experiments reported below, we use a log-linear policy representation, where the parameters correspond to coefficients of features defined over state-action pairs.

Agents acting in an MDP experience the world as a sequence of state-action pairs called a trajectory. We denote a K-length trajectory as $\xi = (s_0, a_0, ..., a_{K-1}, s_K)$ beginning in state s_0 and terminating after K steps. It follows from the definitions above that the probability of generating a K-length trajectory given that the agent executes policy π_θ starting from state s_0 is, $P(\xi|\theta, s_0) = \prod_{t=1}^{K} P(s_{t-1}|s_t, a_{t-1}) P_\pi(a_{t-1}|s_{t-1}, \theta)$. Trajectories are an important part of our query process. They are an intuitive means of communicating policy information. Trajectories have the advantage that the expert need not share a language with the agent. Instead the expert is only required to recognize differences in physical performances presented by the agent. For purposes of generating trajectories we assume that our learner is provided with a strong simulator (or generative model) of the MDP dynamics, which takes as input a start state s, a policy π_θ, and a value K, and outputs a sampled length K trajectory.

In this work, we evaluate policies in an episodic setting where an episode starts by drawing an initial state from P_0 and then executing the policy for a finite horizon T. A policy's value is the expected total reward of an episode. The goal of the learner is to select a policy whose value is close to that of an expert's policy. Note that our work is not limited to finite-horizon problems and can also be applied to infinite-horizon formulations.

In order to learn a policy, the agent presents *trajectory preference queries (TPQs)* to the expert and receives responses back. A TPQ is a pair of length K trajectories (ξ_i, ξ_j)

that originate from a common state s. Typically K will be much smaller than the horizon T, which is important from the perspective of expert usability. Having been provided with a TPQ the expert gives a response y indicating, which trajectory is preferred. Thus, each TPQ results in a training data tuple (ξ_i, ξ_j, y). Intuitively, the preferred trajectory is the one that is most similar to what the expert's policy would have produced from the same starting state. As detailed more in the next section, this is modeled by assuming that the expert has a (noisy) evaluation function $f(.)$ on trajectories and the response is then given by $y = I(f(\xi_i) > f(\xi_j))$ (a binary indicator). We assume that the expert's evaluation function is a function of the observed trajectories and a latent target policy θ^*.

6.2 Bayesian Model and Inference

In this section we first describe a Bayesian model of the expert response process, which will be used to: 1) Infer policies based on expert responses to TPQs, and 2) Guide the active selection of TPQs. Next, we describe a posterior sampling method for this model which is used for both policy inference and TPQ selection.

6.2.1 Expert Response Model

The model for the expert response y given a TPQ (ξ_i, ξ_j) decomposes as follows

$$P(y|(\xi_i, \xi_j), \theta^*)P(\theta^*)$$

where $P(\theta^*)$ is a prior over the latent expert policy, and $P(y|(\xi_i, \xi_j), \theta^*)$ is a response distribution conditioned on the TPQ and expert policy. In our experiments we use an uninformed prior in the form of a Gaussian over θ^* with diagonal covariance, which penalizes policies with large parameter values.

Response Distribution. The conditional response distribution is represented in terms of an *expert evaluation function* $f^*(\xi_i, \xi_j, \theta^*)$, described in detail below, which translates a TPQ and a candidate expert policy θ^* into a measure of preference for trajectory ξ_i over ξ_j. Intuitively, f^* measures the degree to which the policy θ^* agrees with ξ_i relative to ξ_j. To translate the evaluation into an expert response we borrow from previous work (Chu & Ghahramani, 2005). In particular, we assume the expert response is given by the indicator $I(f^*(\xi_i, \xi_j, \theta^*) > \epsilon)$ where $\epsilon \sim N(0, \sigma_r^2)$. The indicator simply returns 1 if the condition is true, indicating ξ_i is preferred, and zero otherwise. It follows that the conditional response distribution is given by:

$$\begin{aligned} P(y = 1|(\xi_i, \xi_j), \theta^*) &= \int_{-\infty}^{+\infty} I(f^*(\xi_i, \xi_j, \theta^*) > \epsilon) N(\epsilon|0, \sigma_r^2) d\epsilon \\ &= \Phi\left(\frac{f^*(\xi_i, \xi_j, \theta^*)}{\sigma_r}\right). \end{aligned}$$

where $\Phi(.)$ denotes the cumulative distribution function of the normal distribution. This formulation allows the expert to err when demonstrated trajectories are difficult to distinguish as measured by the magnitude of the evaluation function f^*. We now describe the evaluation function in more detail.

Evaluation Function. Intuitively the evaluation function must combine distances between the query trajectories and trajectories generated by the latent target policy. We say that a latent policy and query trajectory are in agreement when they produce similar trajectories. The dissimilarity between two trajectories ξ_i and ξ_j is measured by

the trajectory comparison function

$$f(\xi_i, \xi_j) = \sum_{t=0}^{K} k([s_{i,t}, a_{i,t}], [s_{j,t}, a_{j,t}])$$

where the variables $[s_{i,t}, a_{i,t}]$ represent the values of the state-action pair at time step t in trajectory i (similarly for $[s_{j,t}, a_{j,t}]$) and the kernel function k computes distances between state-action pairs. In our experiments, states and actions are represented by real-valued vectors and we use a simple kernel of the form: $k([s, a], [s', a']) = \|s - s'\| + \|a - a'\|$ though other more sophisticated kernels could be easily used in the model.

Given the trajectory comparison function, we now encode a dissimilarity measure between the latent target policy and an observed trajectory ξ_i. To do this let ξ^* be a random variable ranging over length k trajectories generated by target policy θ^* starting in the start state of ξ_i. The dissimilarity measure is given by:

$$d(\xi_i, \theta^*) = E[f(\xi_i, \xi^*)] = \int f(\xi_i, \xi^*) P(\xi^* | \theta^*) d\xi^*$$

This function computes the expected dissimilarity between a query trajectory ξ_i and the K-length trajectories generated by the latent policy from the same initial state.

Finally, the comparison function value $f^*(\xi_i, \xi_j, \theta^*) = d(\xi_j, \theta^*) - d(\xi_i, \theta^*)$ is the difference in computed values between the ith and jth trajectory. Larger values of f^* indicate stronger preferences for trajectory ξ_i.

6.2.2 Posterior Inference

Given the definition of the response model, the prior distribution, and an observed data set $D = \{(\xi_i, \xi_j, y)\}$ of TPQs and responses the posterior distribution is,

$$P(\theta^*|D) \propto P(\theta^*) \prod_{(\xi_i,\xi_j,y)\in D} \Phi(z)^y (1 - \Phi(z))^{1-y},$$

where $z = \frac{d(\xi_j,\theta^*) - d(\xi_i,\theta^*)}{\sigma}$. This posterior distribution does not have a simple closed form and we must approximate it.

We approximate the posterior distribution using a set of posterior samples which we generate using a stochastic simulation algorithm called Hybrid Monte Carlo (HMC) (Duane *et al.*, 1987; Andrieu *et al.*, 2003a). The HMC algorithm is an example of a Markov Chain Monte Carlo (MCMC) algorithm. MCMC algorithms output a sequence of samples from the target distribution. HMC has an advantage in our setting because it introduces auxiliary momentum variables proportional to the gradient of the posterior which guides the sampling process toward the modes of the posterior distribution.

To apply the HMC algorithm we must derive the gradient of the energy function $\bigtriangledown_{\theta^*} \log(P(D|\theta)P(\theta))$ as follows.

$$\frac{\partial}{\partial \theta_i^*} \log[P(\theta^*|D)] = \frac{\partial}{\partial \theta_i^*} \log[P(\theta^*)] + \sum_{(\xi_i,\xi_j,y)\in D} \frac{\partial}{\partial \theta_i^*} \log\left[\Phi(z)^y (1 - \Phi(z))^{1-y}\right]$$

The energy function decomposes into prior and likelihood components. Using our assumption of a Gaussian prior with diagonal covariance on θ^* the partial derivative of the prior component at θ_i^* is

$$\frac{\partial}{\partial \theta_i^*} \log[P(\theta^*)] = -\frac{(\theta_i^* - \mu)}{\sigma^2}.$$

Next, consider the gradient of the data log likelihood,

$$\sum_{(\xi_i,\xi_j,y)\in D} \frac{\partial}{\partial \theta_i^*} \log\left[\Phi(z)^y (1-\Phi(z))^{1-y}\right],$$

which decomposes into $|D|$ components each of which has a value dependent on y.

In what follows we will assume that $y = 1$ (It is straight forward to derive the second case). Recall that the function $\Phi(.)$ is the cumulative distribution function of $N(z; 0, \sigma_r^2)$, Therefore, the gradient of $\log(\Phi(z))$ is,

$$\begin{aligned}\frac{\partial}{\partial \theta_i^*} \log[\Phi(z)] &= \frac{1}{\Phi(z)} \left(\frac{\partial}{\partial \theta_i^*} \Phi(z)\right) = \frac{1}{\Phi(z)} \left(\frac{\partial}{\partial \theta_i^*} z\right) N(z; 0, \sigma_r^2) \\ &= \frac{1}{\Phi(z)} \left(\frac{1}{\sigma_r} \left(\frac{\partial}{\partial \theta_i^*} d(\xi_j, \theta^*) - \frac{\partial}{\partial \theta_i^*} d(\xi_i, \theta^*)\right) N(z; 0, \sigma_r^2)\right).\end{aligned}$$

To complete the derivation recall the definition of $d(\xi, \theta^*)$ from above. The gradient of this function is

$$\begin{aligned}\frac{\partial}{\partial \theta_i^*} d(\xi, \theta^*) &= \int f(\xi, \xi^*) \frac{\partial}{\partial \theta_i^*} P(\xi^*|\theta^*) d\xi^* = \int f(\xi, \xi^*) P(\xi^*|\theta^*) \frac{\partial}{\partial \theta_i^*} \log(P(\xi^*|\theta^*)) d\xi^* \\ &= \int f(\xi, \xi^*) P(\xi^*|\theta^*) \sum_{k=1}^{K} \frac{\partial}{\partial \theta_i^*} \log(P_\pi(a_k|s_k, \theta^*)) d\xi^*.\end{aligned}$$

The final step follows from the definition of the trajectory density which decomposes under the log transformation. For purposes of approximating the gradient this integral must be estimated. We do this by generating N sample trajectories from $P(\xi^*|\theta^*)$ and then compute the Monte-Carlo estimate $-\frac{1}{N} \sum_{l=1}^{N} f(\xi, \xi_l^*) \sum_{k=1}^{K} \frac{\partial}{\partial \theta_i^*} \log(P_\pi(a_k|s_k, \theta^*))$. We leave the definition of $\log(P_\pi(a_k|s_k, \theta^*)$ for the experimental results section where we describe a specific kind of stochastic policy space.

Given this gradient calculation, we can apply HMC in order to sample policy parameter vectors from the posterior distribution. The generated sample can be used for policy selection in a number of ways. For example, a policy could be formed via Bayesian averaging. In our experiments, we select a policy by generating a large set of samples

and then select the sample maximizing the energy function.

6.3 Active Query Selection

Given the ability to perform posterior inference, the question now is how to collect a data set of TPQs and their responses. Unlike many learning problems, there is no natural distribution over TPQs to draw from, and thus, active selection of TPQs is essential. In particular, we want the learner to select TPQs for which the responses will be most useful toward the goal of learning the target policy. This selection problem is difficult due to the high dimensional continuous space of TPQs, where each TPQ is defined by an initial state and two trajectories originating from the state. To help overcome this complexity our algorithm assumes the availability of a distribution $\hat{P}_0$ over candidate start states of TPQs. This distribution is intended to generate start states that are feasible and potentially relevant to a target policy. The distribution may incorporate domain knowledge to rule out unimportant parts of the space (e.g. avoiding states where the bicycle has crashed) or simply specify bounds on each dimension of the state space and generate states uniformly within the bounds. Given this distribution, we consider two approaches to actively generating TPQs for the expert.

6.3.1 Query by Disagreement

Our first approach *Query by Disagreement (QBD)* is similar to the well-known query-by-committee approach to active learning of classifiers (Seung *et al.*, 1992; Freund *et al.*, 1997). The main idea behind the basic query-by-committee approach is to generate a sequence of unlabeled examples from a given distribution and for each example sample

a pair of classifiers from the current posterior. If the sampled classifiers disagree on the class of the example, then the algorithm queries the expert for the class label. This simple approach is often effective and has theoretical guarantees on its efficiency.

We can apply this general idea to select TPQs in a straightforward way. In particular, we generate a sequence of potential initial TPQ states from $\hat{P}_0$ and for each draw two policies θ_i and θ_j from the current posterior distribution $P(\theta^*|D)$. If the policies "disagree" on the state, then a query is posed based on trajectories generated by the policies. Disagreement on an initial state s_0 is measured according to the expected difference between K length trajectories generated by θ_i and θ_j starting at s_0. In particular, the disagreement measure is $g = \int_{(\xi_i,\xi_j)} P(\xi_i|\theta_i, s_0, K)P(\xi_j|\theta_j, s_0, K)f(\xi_i, \xi_j)$, which we estimate via sampling a set of K length trajectories from each policy. If this measure exceeds a threshold then TPQ is generated and given to the expert by running each policy for K steps from the initial state. Otherwise a new initial state is generated. If no query is posed after a specified number of initial states, then the state and policy pair that generated the most disagreement are used to generate the TPQ. We set the threshold t so that $\Phi(t/\sigma_r) = .95$.

This query strategy has the benefit of generating TPQs such that ξ_i and ξ_j are significantly different. This is important from a usability perspective, since making preference judgements between similar trajectories can be difficult for an expert and error prone. In practice we observe that the QBD strategy often generates TPQs based on policy pairs that are from different modes of the distribution, which is an intuitively appealing property.

6.3.2 Expected Belief Change

Another class of active learning approaches for classifiers is more selective than traditional query-by-committee. In particular, they either generate or are given an unlabeled dataset and then use a heuristic to select the most promising example to query from the entire set. Such approaches often outperform less selective approaches such as the traditional query-by-committee. In this same way, our second active learning approach for TPQs attempts to be more selective than the above QBD approach by generating a set of candidate TPQs and heuristically selecting the best among those candidates.

A set of candidate TPQs is generated by first drawing an initial state from from $\hat{P}_0$, sampling a pair of policies from the posterior, and then running the policies for K steps from the initial state. It remains to define the heuristic used to select the TPQ for presentation to the expert.

A truly Bayesian heuristic selection strategy should account for the overall change in belief about the latent target policy after adding a new data point. To represent the difference in posterior beliefs we use the variational distance between posterior based on the current data D and the posterior based on the updated data $D \cup \{(\xi_i, \xi_j, y)\}$.

$$V(P(\theta|D) \parallel P(\theta|D \cup \{(\xi_i, \xi_j, y)\})) = \int |P(\theta|D) - P(\theta|D \cup \{(\xi_i, \xi_j, y)\})| d\theta.$$

By integrating over the entire latent policy space it accounts for the total impact of the query on the agent's beliefs.

The value of the variational distance depends on the response to the TPQ, which is unobserved at query selection time. Therefore, the agent computes the expected

variational distance,

$$H(d) = \sum_{y \in 0,1} P(y|\xi_i, \xi_j, D) V(P(\theta|D) \parallel P(\theta|D \cup \{(\xi_i, \xi_j, y)\})).$$

Where $P(y|\xi_i, \xi_j, D) = \int P(y|\xi_i, \xi_j, \theta^*) P(\theta^*|D) d\theta^*$ is the predictive distribution and is straightforwardly estimated using a set of posterior samples.

Finally, we specify a simple method of estimating the variational distance given a particular response. For this, we re-express the variational distance as an expectation with respect to $P(\theta|D)$,

$$\begin{aligned} V(P(\theta|D) \parallel P(\theta|D \cup d)) &= \int |P(\theta|D) - P(\theta|D \cup d)| \, d\theta = \int \left| P(\theta|D) - P(\theta|D \cup d) \frac{P(\theta|D)}{P(\theta|D)} \right| d\theta \\ &= \int P(\theta|D) \left| 1 - \frac{P(\theta|D \cup d)}{P(\theta|D)} \right| d\theta = \int P(\theta|D) \left| 1 - P(d|\theta) \frac{P(D)}{P(D \cup d)} \right| d\theta \end{aligned}$$

where $P(D)$ and $P(D \cup d)$ are the normalizing constants of the posterior distributions. The final expression is a likelihood weighted estimate of the variational distance. We can estimate this value using Monte-Carlo over a set S of policies sampled from the posterior,

$$V(P(\theta|D) \parallel P(\theta|D \cup (\xi_i, \xi_j, y))) \approx \sum_{\theta \in S} \left| 1 - \frac{P(D)}{P(D \cup d)} P(d|\theta) \right|$$

This leaves the computation of the ratio of normalizing constants $\frac{P(D)}{P(D \cup d)}$ which we estimate using Monte-Carlo based on a sample set of policies from the prior distribution, hence avoiding further posterior sampling.

Our basic strategy of using an information theoretic selection heuristic is similar to early work using Kullback Leibler Divergence ((Cover & Thomas, 1991)) to measure the quality of experiments (Lindley, 1956; Bernardo, 1979). Our approach differs in that we

use a symmetric measure which directly computes differences in probability instead of expected differences in code lengths. The key disadvantage of this form of look-ahead query strategy (shared by other strategies of this kind) is the computational cost.

6.4 Empirical Results

Below we outline our experimental setup and present our empirical results on four standard RL benchmark domains.

6.4.1 Setup

If the posterior distribution focuses mass on the expert policy parameters the expected value of the MAP parameters will converge to the expected value of the expert's policy. Therefore, to examine the speed of convergence to the desired expert policy we report the performance of the MAP policy in the MDP task. We choose the MAP policy, maximizing $P(D|\theta)P(\theta)$, from the sample generated by our HMC routine. The expected return of the selected policy is estimated and reported. Note that no reward information is given to the learner and is used for evaluation only.

We produce an automated expert capable of responding to the queries produced by our agent. The expert knows a target policy, and compares, as described above, the query trajectories generated by the agent to the trajectories generated by the target policy. The expert stochastically produces a response based on its evaluations. Target policies are hand designed and produce near optimal performance in each domain.

In all experiments the agent executes a simple parametric policy,

$$P(a|s,\theta) = \frac{exp(\phi(s)\cdot\theta_a)}{\sum_{b\in A} exp(\phi(s)\cdot\theta_b)}.$$

The function $\phi(s)$ is a set of features derived from the current state s. The complete parameter vector θ is decomposed into components θ_a associated with each action a. The policy is executed by sampling an action from this distribution. The gradient of this action selection policy can be derived straightforwardly and substituted into the gradient of the energy function required by our HMC procedure.

We use the following values for the unspecified model parameters: $\sigma_r^2 = 1$, $\sigma^2 = 2$, $\mu = 0$. The value of K used for TPQ trajectories was set to 10 for each domain except for Bicycle, for which we used $K = 300$. The Bicycle simulator uses a fine time scale, so that even $K = 300$ only corresponds to a few seconds of bike riding, which is quite reasonable for a TPQ.

For purposes of comparison we implement a simple random TPQ selection strategy (Denoted Random in the graphs below). The random strategy draws an initial TPQ state from $\hat{P}_0$ and then generates a trajectory pair by executing two policies drawn i.i.d. from the prior distribution $P(\theta)$. Thus, this approach does not use information about past query responses when selecting TPQs.

Domains. We consider the following benchmark domains.

Acrobot. The acrobot task simulates a two link under-actuated robot. One joint end, the "hands" of the robot, rotates around a fixed point. The mid joint associated with the "hips" attach the upper and lower links of the robot. To change the joint angle between the upper and lower links the agent applies torque at the hip joint. The lower link swings freely. Our expert knows a policy for swinging the acrobot into a

balanced handstand. The acrobot system is defined by four continuous state variables $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$ representing the arrangement of the acrobot's joints and the changing velocities of the joint angles. The acrobot is controlled by a 12 dimensional softmax policy selecting between positive, negative, and zero torque to be applied at the hip joint. The feature vector $\phi(s)$ returns the vector of state variables. The acrobot receives a penalty on each step proportional to the distance between the foot and the target position for the foot.

Mountain Car. The mountain car domain simulates an underpowered vehicle which the agent must drive to the top of a steep hill. The state of the mountain car system is described by the location of the car x, and its velocity v. The goal of the agent controlling the mountain car system is to utilize the hills surrounding the car to generate sufficient energy to escape a basin. Our expert knows a policy for performing this escape. The agent's softmax control policy space has 16 dimensions and selects between positive and negative accelerations of the car. The feature vector $\phi(s)$ returns a polynomial expansion $(x, v, x^2, x^3, xv, x^2v, x^3v, v^2)$ of the state. The agent receives a penalty for every step taken to reach the goal.

Cart Pole. In the cart-pole domain the agent attempts to balance a pole fixed to a movable cart while maintaining the carts location in space. Episodes terminate if the pole falls or the cart leaves its specified boundary. The state space is composed of the cart velocity v, change in cart velocity v', angle of the pole ω, and angular velocity of the pole ω'. The control policy has 12 dimensions and selects the magnitude of the change in velocity (positive or negative) applied to the base of the cart. The feature vector returns the state of the cart-pole. The agent is penalized for pole positions deviating from upright and for movement away from the midpoint.

Bicycle Balancing. Agents in the bicycle balancing task must keep the bicycle bal-

anced for 30000 steps. For our experiments we use the simulator originally introduced in (Randløv & Alstrøm, 1998). The state of the bicycle is defined by four variables $(\omega, \dot{\omega}, \nu, \dot{\nu})$. The variable ω is the angle of the bicycle with respect to vertical, and $\dot{\omega}$ is its angular velocity. The variable ν is the angle of the handlebars with respect to neutral, and $\dot{\nu}$ is the angular velocity. The goal of the agent is to keep the bicycle from falling. Falling occurs when $|\omega| > \pi/15$. We borrow the same implementation used in(Lagoudakis *et al.*, 2003) including the discrete action set, the 20 dimensional feature space, and 100 dimensional policy. The agent selects from a discrete set of five actions. Each discrete action has two components. The first component is the torque applied to the handlebars $T \in (-1, 0, 1)$, and the second component is the displacement of the rider in the saddle $p \in (-.02, 0, .02)$. From these components five action tuples are composed $a \in ((-1, 0), (1, 0), (0, -.02), (0, .02), (0, 0))$. The agent is penalized proportional to the magnitude of ω at each step and receives a fixed penalty for falling.

We report the results of our experiments in Figure 6.2. Each graph gives the results for the TPQ selection strategies Random, Query-by-Disagreement (QBD), and Expected Belief Change (EBC). The average reward versus number of queries is provided for each selection strategy, where curves are averaged over 20 runs of learning.

6.4.2 Experiment Results

In all domains the learning algorithm successfully learns the target policy. This is true independent of the query selection procedure used. As can be seen our algorithm can successfully learn even from queries posed by Random. This demonstrates the effectiveness of our HMC inference approach.

Importantly, in some cases, the active query selection heuristics significantly improve

the rate of convergence compared to Random. The value of the query selection procedures is particularly high in the Mountain Car and Cart Pole domains. In the Mountain Car domain more than 500 Random queries were needed to match the performance of 50 EBC queries. In both of these domains examining the generated query trajectories shows that the Random strategy tended to produce difficult to distinguish trajectory data and later queries tended to resemble earlier queries. This is due to "plateaus" in the policy space which produce nearly identical behaviors. Intuitively, the information content of queries selected by Random decreases rapidly leading to slower convergence. By comparison the selection heuristics ensure that selected queries have high impact on the posterior distribution and exhibit high query diversity.

The benefits of the active selection procedure diminish in the Acrobot and Bicycle domains. In both of these domains active selection performs only slightly better than Random. This is not the first time active selection procedures have shown performance similar to passive methods (Schein & Ungar, 2007). In Acrobot all of the query selection procedure quickly converge to the target policy (only 25 queries are needed for Random to identify the target). Little improvement is possible over this result. Similarly, in the bicycle domain the performance results are difficult to distinguish. We believe this is due to the length of the query trajectories (300) and the importance of the initial state distribution. Most bicycle configurations lead to out of control spirals from which no policy can return the bicycle to balanced. In these configurations inputs from the agent result in small impact on the observed state trajectory making policies difficult to distinguish. To avoid these cases in Bicycle the start state distribution $\hat{P}_0$ only generated initial states close to a balanced configuration. In these configurations poor balancing policies are easily distinguished from better policies and the better policies are not rare. These factors lead Random to be quite effective in this domain.

Finally, comparing the active learning strategies, we see that EBC has a slight advantage over QBD in all domains other than Bicycle. This agrees with prior active learning work, where more selective strategies tend to be superior in practice. The price that EBC pays for the improved performance is in computation time, as it is about an order of magnitude slower.

6.5 Conclusion

This chapter formulates the problem of learning a target policy via trajectory preference queries. Under the developed formulation the agent selects pairs of K-length trajectories, short demonstrations of policy performance, starting from a fixed initial state. The expert expresses a preference by identifying which trajectory best matches the performance of the target policy. To solve this problem this chapter introduces a Bayesian model of the querying process. The model assumes the availability of a prior distribution over the latent target policy space. Additionally, the model allows the expert to err when specifying preferences. It makes the natural assumption that preference specification is more difficult when query trajectories are similar. The Bayesian model naturally supports active methods of selection. For purposes of learning we use Hybrid MCMC (Duane *et al.*, 1987; Andrieu *et al.*, 2003a) to generate samples from the posterior distribution. To use this algorithm we derive the gradient with respect to the latent policy parameters.

This chapter proposes that the agent take direct control over the selection of query data. To exploit the Bayesian model we contribute two simple heuristic methods of query selection. The QBD heuristic selects query policies which produce distinct trajectories. Computing the heuristic is computationally cheap. The EBC heuristic measures the expected impact of the query response on the posterior distribution. Larger values of

the EBC heuristic indicate more impactful queries. Because EBC employs a myopic look-ahead computing EBC is computationally intractable. We introduce a simple method of approximating the EBC heuristic which alleviates some of this burden. We use a simple combinatorial method of optimizing the selection heuristics. Pairs of policies are selected from the posterior sample generated by Hybrid MCMC, and initial states are randomly generated within a constrained space. The optimization algorithm iterates through the candidate queries and selects the best.

To examine the performance of our model, inference algorithm, and query selection heuristics we use four RL benchmark domains. Experiments indicate that our model and inference approach is able to infer quality policies and that the query selection methods are generally more effective than random selection. The EBC heuristic sacrifices computational resources for higher quality queries. Empirically, the performance of active learning using the EBC heuristic is never lower than QBC or random strategies, and EBC outperforms these strategies in both the mountain car and acrobot domains.

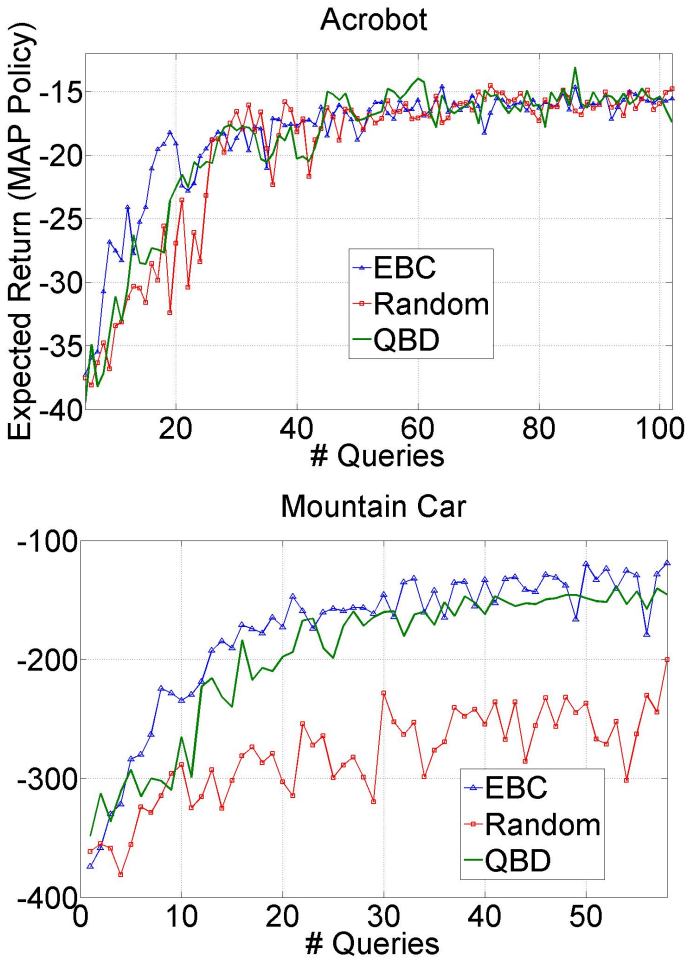


Figure 6.1: Results: We report the expected return of the MAP policy, sampled during Hybrid MCMC simulation of the posterior, as a function of the number of expert queries. Results are averaged over 50 runs. Query trajectory lengths: Acrobot $K = 10$, Mountain-Car $K = 10$

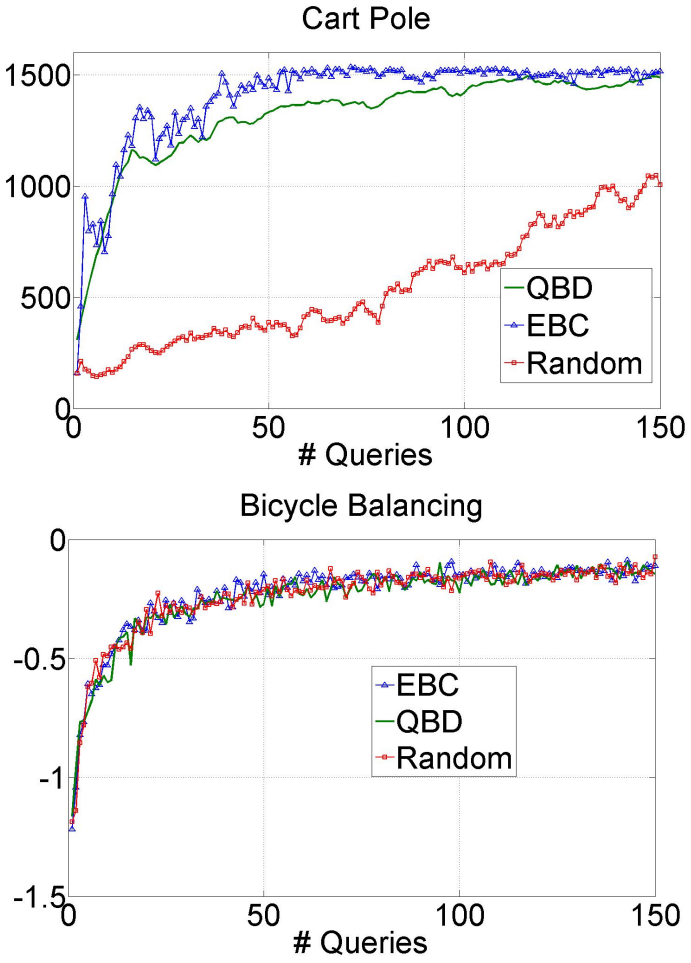


Figure 6.2: Results: We report the expected return of the MAP policy, sampled during Hybrid MCMC simulation of the posterior, as a function of the number of expert queries. Results are averaged over 50 runs. Query trajectory lengths: Cart-Pole $K = 20$, Bicycle Balancing $K = 300$.

Chapter 7: Conclusion

This thesis has introduced Bayesian methods of knowledge representation with the aim of improving exploration in RL. Our contributions include hierarchical models of task relatedness, hierarchical policy priors for multi-agent RL, new models of policy similarity, and a new Bayesian model relating preference data to policies.

In Chapter 3 we generalize model-based Bayesian RL to the MTRL setting. We defined a hierarchical concept of task relatedness that assumes tasks belong to a latent set of classes. It is assumed that tasks belonging to the same class have similar transition and reward functions. The shared class-specific knowledge is stored in a Bayesian prior distribution over the model parameters. We exploit this Bayesian model when exploring in a new task. When a new task generated by the MTRL process belongs to an existing class, our agent exploits this similarity to explore more quickly.

Chapter 4 develops the BPS algorithm for policy search using a Bayesian policy prior. In this chapter we propose that the agents transfer learned policy knowledge between tasks. For purposes of illustration, we defined a hierarchical role-based policy space for multi-agent RL. The policy space assumes that each agent executes a specific role as part of a cooperative joint policy. We use the BPS algorithm to learn from experience the number of roles, the kinds of roles, and how roles should be assigned to agents. We show that BPS can leverage prior experience by transferring roles learned in early tasks to new tasks with similar role structure. Additionally, we show that the BPS algorithm can leverage demonstrations of expert play to learn roles quickly and can use these roles when learning autonomously in new tasks. This chapter introduced a distinct form of

policy knowledge transfer for the MTRL setting.

Chapter 5 introduced a distinct form of generalizing policy knowledge. We present new models of policy similarity for use in BO for RL. The MBOA model exploits imperfect models of the transition and reward functions. It uses these models to compute an approximate surface of the expected return and corrects this surface using a GP model. The GP model is built using the error between the approximate surface and the observed returns data. By generalizing the residual corrections, the MBOA model improves the overall predictive performance. We also propose the Behavior Based Kernel, which introduces a new measure of policy similarity based on the divergence between trajectory densities. This new measure of policy similarity makes explicit use of the sequential nature of the RL problem. We empirically evaluate the new models in a collection of benchmark domains and show that using our models speeds up the BO process.

Finally, in Chapter 6 we introduce a new model of policy learning from trajectory preference queries. We propose that an agent present pairs of trajectories starting from a common state and query an expert for the preferred trajectory. Our Bayesian model assumes that the elicited preferences provide information about a latent target policy. Our agent exploits the Bayesian model to actively learn the target policy by selecting high quality queries. Queries are selected based on maximizing newly developed query evaluation heuristics. We show that our active policy learning algorithm can elicit high quality responses from an expert. In all cases, the active learning strategy performs as well as random query selection, and in most cases the active learning strategy discovers the target policy more quickly. This chapter introduced a new method of learning a posterior distribution over the latent policy space. The principal advantage of this approach is the minimal requirements of the expert. The expert need not be able to perform the desired task, know how to solve it themself, nor be able to provide explicit

action selection suggestions. Instead, the expert must be able to recognize the desired behavior.

References

Akrour, R., Schoenauer, M., & Sebag, M. 2011. Preference-based policy learning. *Pages 12–27 of:* Gunopulos, Dimitrios, Hofmann, Thomas, Malerba, Donato, & Vazirgiannis, Michalis (eds), *Proc. ecml/pkdd'11*. Lecture Notes in Computer Science, vol. 6911. Springer.

Andrieu, Christophe, de Freitas, Nando, Doucet, Arnaud, & Jordan, Michael I. 2003a. An introduction to mcmc for machine learning. *Machine learning*, **50**(1-2), 5–43.

Andrieu, Christophe, de Freitas, Nando, Doucet, Arnaud, & Jordan, Michael I. 2003b. An introduction to mcmc for machine learning. *Machine learning*, **50**(January), 5–43.

Argall, Brenna D., Chernova, Sonia, Veloso, Manuela, & Browning, Brett. 2009. A survey of robot learning from demonstration. *Robot. auton. syst.*, **57**(5), 469–483.

Asmuth, John, Li, Lihong, Littman, Michael L., Nouri, Ali, & Wingate, David. 2009. A bayesian sampling approach to exploration in reinforcement learning. *Pages 19–26 of: Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. UAI '09. Arlington, Virginia, United States: AUAI Press.

Bagnell, J. Andrew (Drew), & Schneider, Jeff. 2003 (August). Covariant policy search. *In: International joint conferences on artificial intelligence*.

Banerjee, B., & Stone, P. 2007. General game learning using knowledge transfer. *In: Proceedings of the 20th international joint conference on artificial intelligence*.

Baxter, Jonathan, Bartlett, Peter L., & Weaver, Lex. 2001. Experiments with infinite-horizon, policy-gradient estimation. *Journal of artificial intelligence research*, **15**(1), 351–381.

Bernardo, J M. 1979. Expected information as expected utility. *Annals of statistics*, **7**(3), 686–690.

Brafman, Ronen I., & Tennenholtz, Moshe. 2003. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. mach. learn. res.*, **3**, 213–231.

Brochu, Eric, Cora, Vlad, & de Freitas, Nando. 2009. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *Technical report tr-2009-023*.

Buşoniu, L., Ernst, D., Schutter, B. De, & Babuška, R. 2011 (Apr.). Approximate reinforcement learning: An overview. *Pages 1–8 of: Proceedings of the 2011 ieee symposium on adaptive dynamic programming and reinforcement learning (adprl 2011)*.

Cheng, Weiwei, Fürnkranz, Johannes, Hüllermeier, Eyke, & Park, Sang-Hyeun. 2011. Preference-based policy iteration: Leveraging preference learning for reinforcement learning. *Pages 312–327 of: Proceedings of the 22nd european conference on machine learning (ecml 2011)*. Springer.

Chu, Wei, & Ghahramani, Zoubin. 2005. Preference learning with gaussian processes. *Pages 137–144 of: Proceedings of the 22nd international conference on machine learning*. ICML '05. New York, NY, USA: ACM.

Cover, Thomas M., & Thomas, Joy A. 1991. *Elements of information theory*. New York, NY, USA: Wiley-Interscience.

Dayan, Peter, & Hinton, Geoffrey E. 1997. Using expectation-maximization for reinforcement learning. *Neural computation*, **9**(February), 271–278.

Dearden, Richard, Friedman, Nir, & Russell, Stuart. 1998. Bayesian q-learning. *Pages 761–768 of: Proceedings of the fifteenth national/tenth conference on artificial intelligence/innovative applications of artificial intelligence*. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Dearden, Richard, Friedman, Nir, & Andre, David. 1999. Model based bayesian exploration. *Pages 150–159 of: Uai*.

Deisenroth, Marc, & Rasmussen, Carl. 2011. Pilco: A model-based and data-efficient approach to policy search. *Pages 465–472 of: Icml*.

Duane, Simon, Kennedy, A. D., Pendleton, Brian J., & Roweth, Duncan. 1987. Hybrid monte carlo. *Physics letters b*, **195**(2), 216 – 222.

Duff, M. 2003. Design for an optimal probe. *In: International conference on machine learning*.

Engel, Y., Mannor, S., & Meir, R. 2003. Bayes meets bellman:the gaussian process approach to temporal difference learning. *In: Icml*.

Engel, Y., Mannor, S., & Meir, R. 2005. Reinforcement learning with gaussian processes. *Pages 201–208 of: International conference on machine learning*.

Ernst, Marc O., & Banks, Martin S. 2002. Humans integrate visual and haptic information in a statistically optimal fashion. *Nature*, **415**(6870), 429–433.

Freund, Yoav, Seung, H. Sebastian, Shamir, Eli, & Tishby, Naftali. 1997. Selective sampling using the query by committee algorithm. *Machine learning*, **28**(2-3), 133–168.

Ghavamzadeh, M., & Engel, Y. 2007a. Bayesian policy gradient algorithms. *In: Nips*.

Ghavamzadeh, Mohammad, & Engel, Yaakov. 2006. Bayesian policy gradient algorithms. *Pages 457–464 of: Nips*.

Ghavamzadeh, Mohammad, & Engel, Yaakov. 2007b. Bayesian actor-critic algorithms. *Pages 297–304 of: Proceedings of the 24th international conference on machine learning*. ICML '07. New York, NY, USA: ACM.

Gordon, Geoffrey. 1999. *Approximate solutions to markov decision processes*. Ph.D. thesis, Robotics Institute, Carnegie Mellon University.

Guestrin, Carlos, Koller, Daphne, Parr, Ronald, & Venkataraman, Shobha. 2003. Efficient solution algorithms for factored mdps. *Journal of artificial intelligence research (jair)*, **19**, 399–468.

Hansen, Nikolaus. 2006. The cma evolution strategy: A comparing review. *Pages 75–102 of:* Lozano, Jose, Larraaga, Pedro, Inza, Iaki, & Bengoetxea, Endika (eds), *Towards a new evolutionary computation*. Studies in Fuzziness and Soft Computing, vol. 192. Springer Berlin / Heidelberg.

Hastings, W. K. 1970. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, **57**(1), 97–109.

Heckerman, David, Geiger, Dan, & Chickering, David M. 1995. Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, **20**(3), 197–243.

Hoffman, Mathew, Doucet, Arnaud, de Freitas, Nando, & Jasra, Ajay. 2007. Bayesian policy learning with trans-dimensional mcmc. *Nips*.

Jaynes, E. T. 2003. *Probability theory: The logic of science*. Cambridge: Cambridge University Press.

Jones, D. R., Perttunen, C. D., & Stuckman, B. E. 1993. Lipschitzian optimization without the lipschitz constant. *J. optim. theory appl.*, **79**(1), 157–181.

Kakade, Sham. 2001. A natural policy gradient. *Pages 1531–1538 of: Nips*.

Kearns, Michael, Mansour, Yishay, & Ng, Andrew Y. 2002. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine learning*, **49**(2-3), 193–208.

Kober, Jens, & Peters, Jan. 2010. Policy search for motor primitives in robotics. *Machine learning*, 1–33.

Konidaris, G., & Barto, A. 2006. Autonomous shaping: knowledge transfer in reinforcement learning. *Pages 489–496 of: Proceedings of the 23rd international conference on machinelearning*.

Körding, K. P., & Wolpert, D. M. 2004. Bayesian integration in sensorimotor learning. *Nature*, **427**(6971), 244–247.

Kotler, Burt P., Ayal, Yoram, & Subach, Aziz. 1994. Effects of predatory risk and resource renewal on the timing of foraging activity in a gerbil community. *Oecologia*, **100**, 391–396.

Lagoudakis, Michail G., Parr, Ronald, & Bartlett, L. 2003. Least-squares policy iteration. *Journal of machine learning research*, **4**.

Lee, Tai Sing S., & Mumford, David. 2003. Hierarchical bayesian inference in the visual cortex. *Journal of the optical society of america. a, optics, image science, and vision*, **20**(7), 1434–1448.

Lindley, D. V. 1956. On a measure of the information provided by an experiment. *The annals of mathematical statistics*, **27**(4), 986–1005.

Lizotte, Daniel. 2008a. *Practical bayesian optimization*. Ph.D. thesis, University of Alberta.

Lizotte, Daniel, Wang, Tao, Bowling, Michael, & Schuurmans, Dale. 2007. Automatic gait optimization with gaussian process regression. *In: Ijcai*.

Lizotte, Daniel James. 2008b. *Practical bayesian optimization*. Ph.D. thesis, Edmonton, Alta., Canada.

MacKay, David J. C. 2002. *Information theory, inference and learning algorithms*. New York, NY, USA: Cambridge University Press.

Mandelik, Yael, Jones, Menna, & Dayan, Tamar. 2003. Structurally complex habitat and sensory adaptations mediate the behavioral responses of a desert rodent to an indirect cue for increased predation risk. *Evolutionary ecology research*, 501 – 515.

Marsella, Stacy, Adibi, Jafar, Al-Onaizan, Yaser, Kaminka, Gal A., Muslea, Ion, & Tambe, Milind. 1999. On being a teammate: experiences acquired in the design of robocup teams. *Pages 221–227 of: International conference on autonomous agents*. Seattle, WA, USA: ACM Press.

Martinson, Eric, & Arkin, Ronald C. 2003. Learning to role-switch in multi-robot systems. *Pages 2727–2734 of: Icra*. IEEE.

Mehta, N., Natarajan, S., Tadepalli, P., & Fern, A. 2005. Transfer in variable-reward hierarchical reinforcement learning. *In: Workshop on transfer learning at nips*.

Metropolis, Nicholas, Rosenbluth, Arianna W., Rosenbluth, Marshall N., Teller, Augusta H., & Teller, Edward. 1953. Equations of state calculations by fast computing machines. *The journal of chemical physics*, **21**(6), 1087–1092.

Meuleau, Nicolas, Peshkin, Leonid, & Kim, Kee-Eung. 2001. Exploration in gradient-based reinforcement learning. *Technical report*.

Mockus, J. 1994. Application of bayesian approach to numerical methods of global and stochastic optimization. *Global optimization*, **4**(4), 347–365.

Moreno, Pedro J., Ho, Purdy P., & Vasconcelos, Nuno. 2004. A kullback-leibler divergence based kernel for svm classification in multimedia applications. *In: Nips*.

Muller, Peter. 1998. Simulation based optimal design. *Bayesian statistics*, **6**, 459–474.

Neal, R. M. 2000. Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, **9**(2), 249–265.

Ng, Andrew Y., & Russell, Stuart J. 2000. Algorithms for inverse reinforcement learning. *Pages 663–670 of: Icml*.

Nguyen, Xuanlong, Wainwright, Martin J., & Jordan, Michael I. 2007. Estimating divergence functionals and the likelihood ratio by penalized convex risk minimization. *In: Advances in neural information processing systems*.

Peters, Jan, Mülling, Katharina, & Altun, Yasemin. 2010. Relative entropy policy search. *In: Aaai*.

Pinsker, M. 1964. *Information and information stability of random variables and processes*. Holden-Day Inc, San Francisco. Translated by Amiel Feinstein.

Poupart, P., Vlassis, N., Hoey, J., & Regan, K. 2006a. An analytic solution to discrete bayesian reinforcement learning. *In: Icml*.

Poupart, Pascal, Vlassis, Nikos, Hoey, Jesse, & Regan, Kevin. 2006b. An analytic solution to discrete bayesian reinforcement learning. *Pages 697–704 of: Proceedings of the 23rd international conference on machine learning*. ICML '06. New York, NY, USA: ACM.

Price, Bob, & Boutilier, Craig. 2003. Accelerating reinforcement learning through implicit imitation. *J. artif. intell. res. (jair)*, **19**, 569–629.

Randløv, Jette, & Alstrøm, Preben. 1998. Learning to drive a bicycle using reinforcement learning and shaping. *Pages 463–471 of: Icml*.

Rasmussen, C. E., & Ghahramani, Z. 2002. Infinite mixtures of gaussian process experts. *Pages 881–888 of: Nips*.

Rasmussen, C. E., & Kuss, M. 2004. Gaussian processes in reinforcement learning. *In: Nips*.

Rasmussen, Carl Edward, & Williams, Christopher K. I. 2005. *Gaussian processes for machine learning (adaptive computation and machine learning)*. The MIT Press.

Reid, Mark D., & Williamson, Robert C. 2009. Generalised pinsker inequalities. *In: Colt*.

Ross, Stéphane, & Pineau, Joelle. 2008. Model-based bayesian reinforcement learning in large structured domains. *Pages 476–483 of: Uai*.

Ross, Stéphane, Chaib-draa, Brahim, & Pineau, Joelle. 2008. Bayesian reinforcement learning in continuous pomdps with application to robot navigation. *Pages 2845–2851 of: Icra*.

Schaal, Stefan. 1996. Learning from demonstration. *Pages 1040–1046 of: Nips*.

Schein, Andrew I., & Ungar, Lyle H. 2007. Active learning for logistic regression: an evaluation. *Mach. learn.*, **68**(3), 235–265.

Seung, H. S., Opper, M., & Sompolinsky, H. 1992. Query by committee. *Pages 287–294 of: Proceedings of the fifth annual workshop on computational learning theory*. COLT '92. New York, NY, USA: ACM.

Shelton, Christian Robert. 2001 (Aug). *Importance sampling for reinforcement learning with multiple objectives*. Ph.D. thesis, MIT.

Shi, Lei, & Griffiths, Thomas L. 2009. Neural implementation of hierarchical bayesian inference by importance sampling. *Pages 1669–1677 of: Nips*.

Sorg, Jonathan, Singh, Satinder P., & Lewis, Richard L. 2010. Variance-based rewards for approximate bayesian reinforcement learning. *Pages 564–571 of: Uai.*

Stocker, Alan, & Simoncelli, Eero P. 2007. A bayesian model of conditioned perception. *In: Nips.*

Stone, Peter, & Veloso, Manuela. 1999. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial intelligence.*

Strens, Malcolm J. A. 2000. A bayesian framework for reinforcement learning. *Pages 943–950 of: Proceedings of the seventeenth international conference on machine learning.* ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Sullivan, Heather L., Curtin, Charles G., Reynolds, Caryn A., & Cardiff, Scott G. 2001. The effect of topography on the foraging costs of heteromyid rodents. *Journal of arid environments*, **48**(3), 255 – 266.

Sutton, Richard, & Barto, Andrew. 1998. *Reinforcement learning:an introduction.* MIT Press.

Thompson, W. R. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, **25**, 285–294.

Wang, Tao, Lizotte, Daniel, Bowling, Michael, & Schuurmans, Dale. 2005. Bayesian sparse sampling for on-line reward optimization. *Pages 956–963 of: Proceedings of the 22nd international conference on machine learning.* ICML '05. New York, NY, USA: ACM.

Watkins, Christopher J. C. H., & Dayan, Peter. 1992. Q-learning. *Machine learning*, **8**(3-4), 279–292.

Williams, C. K. I., & Rasmussen, C. E. 1996. Gaussian processes for regression. *In: Neural information processing systems.*

Williams, Ronald J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, **8**, 229–256.

Wilson, Aaron, Fern, Alan, Ray, Soumya, & Tadepalli, Prasad. 2007. Multi-task reinforcement learning: a hierarchical bayesian approach. *Pages 1015–1022 of: Proceedings of the 24th international conference on machine learning.* ICML '07. ACM.