Model-based Bayesian Reinforcement Learning with Generalized Priors

BY John Thomas Asmuth

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Michael L. Littman

and approved by

New Brunswick, New Jersey

May, 2013

ABSTRACT OF THE DISSERTATION

Model-based Bayesian Reinforcement Learning with Generalized Priors

by John Thomas Asmuth

Dissertation Director: Michael L. Littman

Effectively leveraging model structure in reinforcement learning is a difficult task, but failure to do so can result in computer agents that repeatedly take sub-optimal actions, despite having enough information to perform better. The Bayesian approach is a principled and well-studied method for leveraging model structure, and it is useful to use in the reinforcement learning setting.

This dissertation studies different methods for bringing the Bayesian approach to bear for model-based reinforcement learning agents, as well as different models that can be used. The contributions include several examples of models that can be used for learning MDPs, and two novel algorithms, and their analyses, for using those models for efficient exploration: BOSS and BFS3.

The Bayesian approach to model-based reinforcement learning provides a principled method for incorporating prior knowledge into the design of an agent, and allows the designer to separate the problems of planning, learning and exploration. The BOSS and BFS3 algorithms are efficient (polynomial time) mechanisms for decision making within this framework with provable bounds on their accuracy.

Preface

Portions of this dissertation are based on work previously published or submitted for publication by the author [1, 2].

Acknowledgements

The journey to a successful Ph.D. is not one that can be made alone. Many people have my special thanks, including:

1. My advisor, Michael Littman, brought me into the world of reinforcement learning when I was still an undergrad. He shepherded me into the Computer Science graduate program at Rutgers, and mentored me throughout my many years as a member of the Rutgers Laboratory for Real Life Reinforcement Learning. His insight and enthusiasm provided the lion's share of my education as a graduate student, and for that he deserves special thanks.

2. Jason Williams and Suhrid Balakrishnan mentored me during an internship at AT&T Research, where I was able to solidify much of my current understanding of Bayesian modeling and non-parametrics. Their mentorship was invaluable, and they have my thanks.

3. All my co-authors that have yet to be mentioned—Lihong Li, Ali Nouri, David Weissman, David Wingate, Haakan Younes, and Robert Zinkov—have helped push my research forward in more ways than one, and they have my thanks.

4. Over the years, many people have come and gone through the lab, and I consider them all my friends and colleagues. They are Monica Babes-Vroman, Erick Chastain Carlos Diuk, Sergiu Goschin, Bethany Leffler,

Lihong Li, Chris Mansely, Vukosi Marivate, Ali Nouri, Alex Strehl, Tom Walsh, Ari Weinstein, and Michael Wunder. I valued our time as colleagues, and thank you all for the priviledge and the lunches.

5. My Qualifying Exam committee, Michael Littman, Vladimir Pavlovich, Chung-Chieh Shan, and Vinod Ganapathy, provided excellent feedback at that early stage of my research, and they have my thanks.

6. My Ph.D. Defense committee, Michael Littman, Vladimir Pavlovich, Haym Hirsh, provided incalculably valuable help, during my pre-defense and defense, with creating the best dissertation possible, and they have my thanks.

7. My parents, Jane and Charles Asmuth, whose example and and high esteem for academic achievement set the stage for my own intellectual and academic growth.

Dedication

This dissertation is dedicated to my wonderful wife Stephanie, for putting up with my graduate student income for all these years, and to our future life together.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

One of the fundamental goals for researchers working in the field of artificial intelligence is to make computers act more like "us". That is, we can play chess, or drive a car, or control a robot, so why is it so difficult to make a computer program to do things the way *we* do? Unfortunately, human behavior appears to arise from an arcane mix of intuition, acquired knowledge, and calculation. Automated problem solvers have gotten very good at the calculation aspects, but effectively incorporating background knowledge into the agent's calculations remains difficult.

The Bayesian approach to machine-learning attempts to address the knowledge issue. With this approach, knowledge given to the algorithm before any acting occurs or from its own related experiences is formalized into a *prior*. The prior describes what the algorithm designer thinks the world "might be". As a result, priors can be either extremely vague or even uninformative, but they can also be exact or nearly exact descriptions of the world. The Bayesian approach to machine-learning takes the prior and combines it with observations made from the world to create a *posterior*, or the agent's better-informed impression of the environment in which it acts.

In this dissertation, I discuss methods of applying the Bayesian approach to building and using models for sequential decision making. Using a prior to shape an agent's concept of how its environment works can greatly reduce

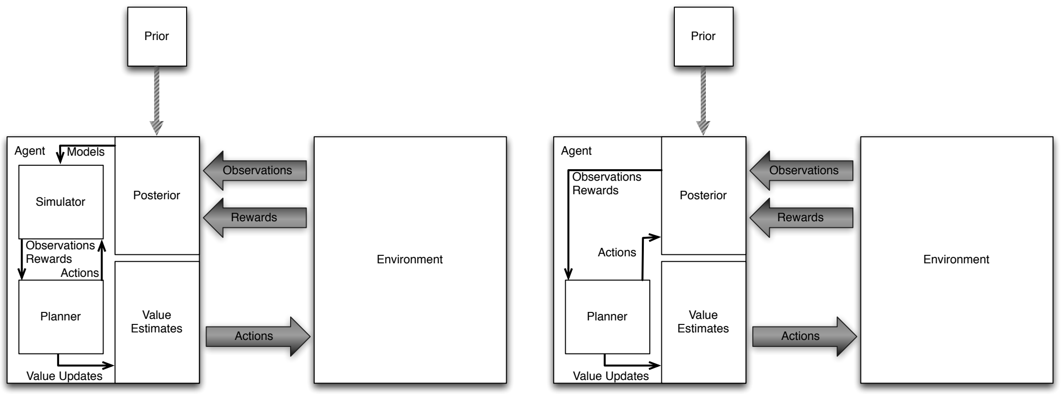

Figure 1.1: The two different approaches to Bayesian model-based reinforcement learning that are discussed in this dissertation are the model-sampling approach (left) and the experience-sampling approach (right). With a model-sampling approach, the agent will sample MDPs from the posterior and use those MDPs to run its internal simulation that it uses to create an estimated value function. With an experience-sampling approach, the agent treats the posterior itself as an MDP, and samples observations and rewards directly from the posterior. The two main algorithmic contributions of this dissertation, **BOSS** and **BFS3**, use alternate approaches, with **BOSS** being a model sampler and **BFS3** being an experience sampler.

the amount of data it must collect before performing well; if you tell it that something can or can't happen a certain way, it won't need to test those things to find out.

The approach I use here to sequential decision making and reinforcement learning can be divided into distinct blocks: modeling and planning. Figure 1.1 illustrates the separation of these components. By drawing a strong distinction between the two areas of agent design, one can more easily draw upon the results of their respective communities. The Bayesian machine-learning community focuses its efforts on building complex models that can be used as priors in learning, and works on efficient inference approximations. The planning community focuses its research on efficient ways to act, once given a model. It is only natural to seek to try to turn the fruits of these two communities into a pipeline for learning and sequential decision making.

It is important to note that the best pipeline in this situation is rarely the greedy one, where the agent infers a model, feeds it directly into the planner and follows the planner's advice. In Chapters 3, 4, and 5, I will describe, motivate and analyze several innovative methods for combining the results of model inference with different sorts of planners.

The particular flavor of learning and sequential decision making problems approached here is called reinforcement-learning. Reinforcement learning provides a flexible way to evaluate a possible behavior through the use of numerical rewards that the agent seeks to maximize. Simple "find the shortest path to the goal" behavior, that much of planning research is centered around, is insufficient for many more realistic scenarios where there can be many factors to consider when comparing the "goodness" of two agent behaviors.

1.1 Reinforcement Learning

Reinforcement learning, or RL, is a framework for problems in which an agent faces both an unknown world and/or an unknown goal. In a reinforcement-learning scenario, the agent interacts with the environment by making observations and performing actions. Each time an action is performed, a new observation is returned to the agent along with a numerical reward signal. The agent's goal is, then, to maximize the cumulative reward over the course of the experiment.

Reinforcement learning model to any problem that has an agent, an environment and rewards. Shortest-path problems can be represented by a reward function that gives -1 for each step before the goal, and 0 or some positive reward for every step after the goal is reached. Bandit problems, a class of reinforcement-learning problems in which the actions taken in the past have no effect on actions taken in the future (though the previous results can affect the agent's *perception* of what might happen in the future), can be mapped to online advertizing, where the agent is trying to figure out what ads a typical user is most likely to click.

The related fields of planning and control theory, while initially appearing to solve many of the same kinds of problems, address only part of the story. Planning and control theory are two ways to figure out what to do in a known-model situation. That is, the situation where you have a complete understanding of the world in which your agent exists, and there is no need to learn about or explore the environment. The field of reinforcement-learning makes the opposite assumption: that some portion of the world must be learned to develop good agent behavior.

With an unknown world, there is a tension between acting to optimize reward based on what the agent already knows, and taking actions that are very likely not the right thing to do, in order to learn more about the world so that the agent can gather reward more efficiently in the future. This tension is referred to as the *exploration/exploitation trade-off*, and is central to the field of reinforcement-learning.

Since the observations made by the agent often correspond strongly with the previous observation and the action performed, reinforcement-learning shares some attributes with active learning. The agent can direct the kinds of data it receives by deciding which action to perform, but it is also limited by the previous observation.

Since it is often difficult to distinguish between two infinite sums, RL researchers lean on the idea of a discount factor, usually labeled γ. An agent's optimal policy is then the one that maximizes the expected cumulative discounted sum of rewards. This sum is called the expected return. The expected return for a given policy π is denoted R_π:

$$R_\pi = \sum_{t=0}^{\infty} \gamma^t E_\pi[r_t]. \tag{1.1}$$

The optimal policy, π^*, is the policy that maximizes the return:

$$\pi^* = \operatorname*{argmax}_{\pi} R_\pi. \tag{1.2}$$

1.1.1 Markov Decision Processes

A common way to concisely describe a reinforcement-learning environment is the Markov Decision Process, or MDP. With an MDP, the observation is a complete description of the configuration, or state, of the world, and the next observed state is a function only of the previous state and action performed. That is, if an agent begins in state s_1 and performs action a_1, the odds that it ends up in state s_2 are the same no matter when this occurs — the next-state distribution is unchanging. It is important to note that the agent's *impression* of the next-state distribution is free to change, but the actual distribution, which may be unknown, is stationary.

An MDP is formally defined as the tuple $\langle S, A, R, T \rangle$, where S is the set of states, A is the set of actions, $R : S \times A \rightarrow \Re$ is the reward function, and $T : S \times A \rightarrow \Pi(S)$ is the transition function. The state and action sets S and A may be a discrete collection or a continuous range. The reward function R maps state-action pairs to numerical rewards, and the transition T maps state-action pairs to distributions over next-states. Usually, S and A are given, and

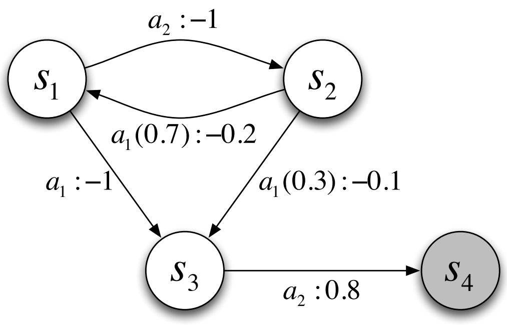

Figure 1.2: an MDP

R and T begin unknown and must be learned.

An MDP is often represented by a graph (for example, see Figure 1.2), where nodes are states and edges are actions. At any given time, the agent is said to be *in* some state, and can *take* an action. As a result of taking the action from the state, the agent will receive a numerical reward and end up in another state, or possibly stay in the same state.

The expected return of the optimal policy for the MDP, starting in a particular state, is called the *value* of that state. The value of a state is denoted by $V(s)$, and is captured by a recurrence relation. The expected return for being in a particular state, taking a given action, and following the optimal policy thereafter, is called the *Q-value* of that state-action pair. The Q-value of a state-action pair is denoted by $Q(s, a)$. The functions Q and V can be related by:

$$V(s) = \max_a Q(s, a), \tag{1.3}$$

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a)(s')V(s'). \tag{1.4}$$

Using the definition of $V(s)$, we know that the optimal policy is the one that always chooses the action with the highest Q-values for the current state.

1.1.2 Planning

The act of finding an optimal or near-optimal policy for a known MDP is called *planning*. Some planning techniques will be given a more thorough treatment in Chapter 3. This chapter will briefly describe some simple planning techniques.

Value Iteration

Value iteration, outlined in Algorithm 1, is a planning method which applies the idea of dynamic programming to solving an MDP with a discrete state space and discrete action space. The values that are iterated are the Q-values. The Q-values are given some initial guess (for instance, zero), and Equations 1.3 and 1.4 are applied iteratively to all state-action pairs (for the Q-value) and to all states (for the value).

Once the difference between successive estimates of the Q-values, measured by the ∞-norm[1], is smaller than $(1-\gamma)\epsilon/2$, the difference between the last estimate and the truth is guaranteed to be no larger than ϵ. Eventually, the error will shrink to an acceptably small level, and an approximately optimal policy will be found.

For manageably small numbers of states and actions, value iteration is an effective and easy method for planning. Because the minimum accuracy ϵ is specified, the algorithm is gauranteed to finish after a finite number of "sweeps" of updates over the states.

[1] The infinity norm of a vector is the absolute value of its largest element.

```
Input: State set $S$, action set $A$, reward function $R$, transition function $T$,
       discount factor $\gamma$, accuracy $\epsilon$
Output: Q-value table $Q$
$\forall_s \; V_0(s) \leftarrow 0$
$\forall_{s,a} \; Q_0(s,a) \leftarrow 0$
$e_0 \leftarrow \infty$
$i \leftarrow 0$
while $e_i > \epsilon$ do
    $i \leftarrow i + 1$
    $e_i \leftarrow 0$
    for $s \in S$ do
        for $a \in A$ do
            $Q_i(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} T(s'|s,a) V_{i-1}(s')$
        $V_i(s) \leftarrow \max_a Q_i(s,a)$
        $e_i \leftarrow max(e_i, |V_i(s) - V_{i-1}(s)|)$
return $Q_i$
```

Algorithm 1: Value iteration$(S, A, R, T, \gamma, \epsilon)$

1.1.3 Outcomes

It is often useful to think of transitions from state to state in terms of the *outcome* rather than the resulting state [4, 5].In the context of this dissertation, an outcome is a summary of a change from one state to another. An example in a grid-world is where each state is assigned a coordinate specifying a cell in the grid. With the origin at the bottom left, moving from the cell $(4, 2)$ to the cell $(4, 3)$ can be described by starting in the cell $(4, 2)$ and moving "up" $(0, 1)$. In this situation, the outcome is "up" and it describes how to derive the destination state given the beginning state.

More formally, we have a set of states S, a set of outcomes O, an outcome function $f : S \times O \to S$, and an inverse outcome function $f^{-1} : S \times S \to O$. In the grid-world example, S is the set of all possible cell coordinates, and O is the set $\{\text{up}, \text{down}, \text{left}, \text{right}, \text{stay}\}$. f is a function that takes a state (for instance, $(4, 2)$) and an outcome ("up") and produces the next state ($(4, 3)$). The inverse outcome function takes two states and returns the outcome that describes the transition from the first state to the second state if it exists.

The concept of an outcome becomes especially useful when trying to apply the same dynamics to two or more different states. In the grid-world example, it is very likely that every state-action pair has a unique next-state distribution. But, observations on outcomes made in one state can very often be applied successfully to learning another state.

Chapter 2 discusses some ways in which outcome distributions can be used to cluster states together, and how these groupings can be used to speed learning and drive exploration.

1.1.4 Optimality Guarantees

A decision-making algorithm's policy is said to be *optimal* if it maximizes the expected return. Since it is generally impossible to infer an optimal policy before any learning has taken place, weaker guarantees are necessary.

Convergence

A reinforcement-learning algorithm has a *convergence guarantee* if it will provably act with the optimal policy in the limit, or after an infinite number of steps in the environment. While this guarantee is fairly weak, theoretically, algorithms that assert it often converge to something near-optimal in a much shorter period of time.

Q–learning [6] is an example of a reinforcement-learning algorithm with a convergence guarantee.

PAC-MDP

Probably Approximately Correct in MDPs, or PAC-MDP, is a guarantee that asserts near-optimality with high likelihood in a "short" amount of "time". An algorithm that is PAC-MDP divides its interactions with the environment into two catagories: exploration steps and exploitation steps. All exploitation steps must have near-optimal actions, or those whose Q–values are within a given accuracy threshold of the state's value, denoted by ϵ. For an algorithm to be PAC-MDP, the number of exploration steps may not exceed a polynomial of the parameters of the environment (often the parameters will be the number of states and actions available in the environment). The algorithm is allowed some overall chance of failure, denoted by δ.

An algorithm that is PAC-MDP will, with high probability, make approximately optimal decisions for all but a polynomial number of steps, where the polynomial is a function of the number of states, actions, ϵ and δ.

RMAX [3] is an example of a reinforcement-learning algorithm with a PAC-MDP guarantee.

Bayes-optimal

Bayes-optimality is an optimality guarantee made in the context of an MDP prior. The prior distribution acts as knowledge about the environment given to the agent before learning occurs. If the MDP truly is drawn from the provided prior, then the policy that acquires the highest expected return is well-defined, if difficult to compute.

Since every step in the environment affects the agent's belief about the environment in a well-defined way, it is possible to account for all possible step sequences and belief updates over some finite horizon.

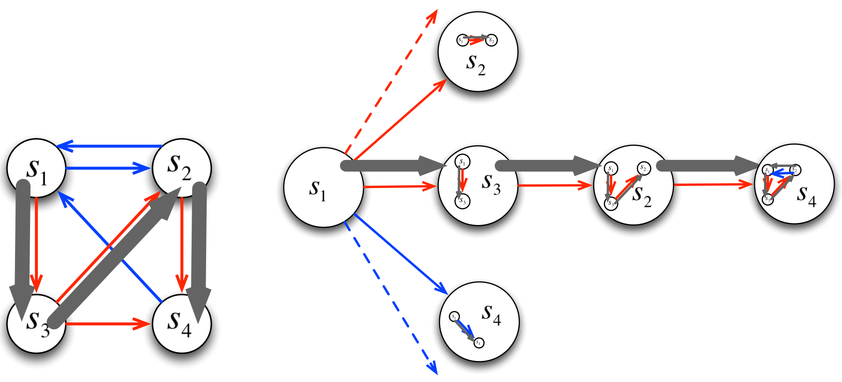

Figure 1.3: **Left:** an agent makes a trajectory through an MDP, starting at s_1 and following the grey arrows. **Right:** the agent's trajectory through the corresponding BAMDP, where each belief-state has information about what has happened before, as well as the concrete state. Note that once the agent reaches a particular belief-state in the BAMDP, many belief states become unreachable since their internal beliefs are inconsistent with the experience that has been realized.

Bayesian learning in an MDP can be represented by a Bayes-Adaptive MDP, or BAMDP [7]. A BAMDP is a special MDP laid out like a directed acyclic graph whose states are belief-states and whose actions are the actions that can be performed in the actual MDP. The belief-state expresses the agent's knowledge about the environment and a concrete state in which the agent can exist. The knowledge of the environment can be expressed by the combination of the prior and the observations made up to that point, or the posterior.

Figure 1.3 shows a brief MDP traversal and the equivalent traversal in the corresponding BAMDP. Inside each state in the BAMDP, which I will call a *belief-state*, there is a pairing of a *concrete-state* from the real MDP and the history leading to that belief-state. The transition probabilities from belief-states in the BAMDP are known functions of the belief-state's posterior.

The intuition behind the accuracy of the BAMDP for Bayesian-optimal decision making follows. Given some current belief-state for the agent, there is some set of next belief-states that can occur (this set may be infinite or continuous) for each action. The probability of each possible next belief-state is well-defined, and the posterior represented by the belief-state is the combination of the previous posterior and the step that could theoretically have occurred: each of the next belief-states represent one possible world that could result from taking one of the actions.

With a model prior ϕ that supports MDPs with state-space S, action-space A, and discount factor γ, the corresponding BAMDP is a new MDP with the same action-space and discount factor, but a new state-space. The state-space is $S \times H$, where H is the set of all possible sequences of transitions, where a transition is a tuple (s, a, s', r). The prior ϕ can then be conditioned on a history h to get the posterior $\phi|h$. The transition function of the BAMDP $M|\phi$ is then defined to be

$$P(\langle s', h + (s, a, s', r)\rangle | \langle s, h\rangle, a) \quad = \quad \int_M P(s', r|s, a, M)\phi(M|h)dM. \qquad (1.5)$$

In Equation 1.5, the transition likelihood to the concrete state in question, s', is considered for all possible MDPs M and weighted according to their posterior likelihoods, taken from $\phi|h$.

Finding the value of a belief-state can then be done recursively via dynamic programming, treating the set of next belief-states as a set of independent planning problems solved in the exact same way. When the agent takes a step in the actual environment, the agent's new belief matches exactly one of the beliefs considered in the planning problem from the previous state.

If the environment is truly drawn from the agent's prior, simulating experience according to the prior and calculated posteriors is the exact same process as running through a single trajectory in the experiment. In fact, the entire BAMDP is known perfectly. Unfortunately, it is usually difficult to completely solve, since the complete BAMDP has an infinite number of states. Even if we consider only those states reachable in a fixed number of steps (sufficient for approximate planning when there is a discount factor), the number of reachable BAMDP belief-states grows exponentially with that horizon length, the number of actions, and the number of different concrete states reachable in a given transition.

PAC-BAMDP

Probably Approximately Correct for Bayes Adaptive MDPs (also known as near Bayes-optimal [8]), or PAC-BAMDP [9], is an optimality guarantee for Bayesian reinforcement-learning.

Exact Bayes-optimal policy inference is generally intractable. Even with a small number of discrete actions and discrete states with a finite horizon, the size of the tree grows exponentially with its depth (with A actions, S states, and a depth of d, there can be up to $(AS)^d$ nodes visited by any algorithm. This growth rate means that even if we limit our planning to some depth[2], fully evaluating every possibility is very expensive.

As is common in machine-learning tasks, we can opt to satisfy ourselves with a probably approximately correct variation. Instead of considering all possible next belief-states in the search tree, we may consider a sufficiently representative subset. Limiting ourselves to the smaller set introduces some probability of failure, denoted δ. In general, as δ grows small, the necessary size of the subset grows large. Instead of searching infinitely deep into the tree, we may consider nodes only up to a certain depth. Searching only to

[2] With a discount factor γ, an accuracy constraint ϵ, and a maximum value $V_{\max}$, we need not consider anything farther than $\log_\gamma \frac{\epsilon}{V_{\max}}$ steps away (see the Bounded Horizon Lemma for details).

some depth introduces an accuracy error, denoted ϵ. In general, as ϵ goes to zero, the necessary depth of the search tree goes to infinity.

An algorithm that is PAC-BAMDP will, with probability $1 - \delta$, make ϵ-Bayes-optimal decisions for all but a polynomial number of steps, where the polynomial is a function of the number of concrete states (not the number of belief-states), actions, ϵ, $1/(1 - \gamma)$, and δ.

It is important to understand exactly what is entailed when an algorithm is PAC-BAMDP. Starting from some particular belief state, there is no guarantee that the discounted return seen from that belief state is optimal or Bayes-optimal. In fact, the agent is able to choose very poor actions with extremely poor returns. Even if only the first action from some belief state is sub-optimal, if it brings the agent to a part of the state space with poor rewards, the return that results from that first action can be arbitrarily worse than the return associated with the best action. This example, where one sup-optimal action is followed by many optimal or near-optimal actions, is in line with the definition and spirit of the PAC-BAMDP guarantee: the total number of actions that are not near-optimal is small, even if those near-optimal actions are especially bad.

Since a single sub-optimal action can have a severe effect on the return seen from that belief-state onwards, what can be said about the overall returns during the lifetime of a PAC-BAMDP agent? First, we can define regret.

Definition 1. *Regret is the difference between the agent's expected discounted future cumulative reward, when beginning at a particular belief state, and the expected discounted future cumulative reward that would result from following the Bayes-optimal policy on every step, starting from that same belief state.*

Then, we can define the expected regret.

Definition 2. *Expected regret is the average regret of all belief states in the sequence of belief states visited by an agent over its lifetime.*

An expected regret of zero indicates that the agent's policy was exactly Bayes-optimal, and an expected regret of ϵ means that the policy was close to Bayes-optimal, on average. An expected regret of ϵ does not mean that the agent's policy was ϵ-Bayes-optimal; several actions could have been worse than ϵ-Bayes-optimal, as long as there were others that were closer to exactly Bayes-optimal to make up for the shortfall. With a long enough sequence of belief states, an agent that has a bounded number of steps that are *not* ϵ-Bayes-optimal will have an expected regret that approaches something no more than ϵ. A PAC-BAMDP algorithm does exactly that: With high probability, it bounds the number of steps where the agent chooses an action that is not ϵ-Bayes-optimal, and since the bound is small, the expected regret approaches ϵ more quickly than an algorithm with a larger bound on the number of sub-Bayes-optimal steps. A lower expected regret is good, because it indicates that the agent is making more ϵ-optimal decisions.

The concept of PAC-BAMDP is not quite directly analogous to PAC-MDP. With a PAC-MDP algorithm, it is normally impossible to act optimally for all steps. That is, the polynomial number of mistakes is necessary to find the optimal policy later — there is simply not enough information available to the agent until many potentially sub-optimal steps are taken. With PAC-BAMDP, it is possible to achieve ϵ-δ Bayes-optimal behavior with *zero* sub-optimal steps. All information needed to act Bayes-optimally is contained in the prior.

An example of an ϵ-δ-Bayes optimal algorithm that requires zero sub-optimal steps is the application of **Sparse Sampling** to the BAMDP. The **Sparse Sampling** algorithm, discussed in detail in Chapter 3, performs an exhaustive tree search up to some provided depth d, trying every action from each visited state C times in order to get a good representation of the next-state distribution. The quantities d and C are chosen to ensure that a ϵ-accurate value for the state is found with probability $1 - \delta$. Given d and C, **Sparse Sampling** will have to create $(AC)^d$ nodes in its search tree, where A is the number of actions. As a result of this exponential component to the algorithm's runtime, and the fact that C and d need to be phenomenally large[3] to ensure a reasonable accuracy, **Sparse Sampling** is impractical to use even in most trivial problems.

PAC-BAMDP can then be called a compromise between a reinforcement-learning algorithm's sample complexity and computation complexity. **Sparse Sampling** on the BAMDP has a sample complexity of 0, but an exponential computation complexity. In contrast is **Bayesian DP**, presented in Chapter 3, which has low computational complexity, but potentially unbounded sample complexity. Chapter 3, on related work, and Chapters 4 and 5, on the contributions of this dissertation, describe algorithms with PAC-BAMDP guarantees and several ways to approach the more traditional computation complexity issue.

The PAC-MDP Theorem

This section presents a framework for proving that an algorithm offers a PAC-MDP guarantee.

This proof is an adaptation of earlier work [10, 11] with one minor change: instead of saying a particular state-action pair is *known* when it has been tried

[3] The quantity C must be large enough so that all nodes in the search tree have enough children to be able to estimate their dynamics properly. However, the number of nodes in the search tree grows with $(AC)^d$. This feedback loop results in bounds for C that are astoundingly high.

at least some threshold number of times, a state-action pair is said to be *known* when other state-action pairs of the same *class* have been visited at least some threshold number of times. Introducing the concept of a *class* of state-action pairs allows the analysis to work smoothly with BAMDPs, which have an infinite number of states. Though the number of states may be infinite, we can choose *classes* for each state-action pair that identify what part of that pair is being learned about (that is, we can ignore history when choosing a *class*).

Bounded Horizon Lemma. Let $V_m^\pi(s, H)$ be the expected value of following policy π in MDP m for H steps, starting at state s. Then,

$$V_m^\pi(s, H) \;\geq\; V_m^\pi(s) - \epsilon_H, \tag{1.6}$$

when

$$H \;=\; \log_\gamma(\epsilon_H / V_{\max}). \tag{1.7}$$

Proof. Let the return R be the discounted sum of rewards from time-step 0 onwards, and $R(H)$ be the discounted sum of rewards for the first H steps. That is,

$$R \;=\; \sum_{t=0}^{\infty} \gamma^t r_t, \tag{1.8}$$

$$R(H) \;=\; \sum_{t=0}^{H-1} \gamma^t r_t. \tag{1.9}$$

Assuming without loss of generality that $0 \leq r_t \leq R_{\max}$, we can bound the difference between R and $R(H)$:

$$\begin{aligned} R - R(H) &= \sum_{t=H}^{\infty} \gamma^t r_t, && (1.10)\\ &\leq \sum_{t=H}^{\infty} \gamma^t R_{\max}, && (1.11)\\ &\leq \frac{\gamma^H}{1-\gamma} R_{\max}, && (1.12)\\ &\leq \gamma^H V_{\max}. && (1.13) \end{aligned}$$

Therefore, to bound the difference at ϵ_H, we must choose an H such that

$$\epsilon_H = \gamma^H V_{\max}, \tag{1.14}$$

$$H = \log_\gamma \left(\frac{\epsilon}{V_{\max}} \right). \tag{1.15}$$

If choosing H this way bounds the difference between any sequence of returns and its H horizon to ϵ_H, it must also bound the expected return under any policy. □

Simulation Lemma [11, 12]. Let m_1 and m_2 be two MDPs, such that

$$\forall_{s,a,s'} |T_1(s'|s,a) - T_2(s'|s,a)| \leq \epsilon_T, \tag{1.16}$$

$$\forall_{s,a} |R_1(s,a) - R_2(s,a)| \leq \epsilon_R. \tag{1.17}$$

Then,

$$\forall_s |V_1(s) - V_2(s)| \leq s(\epsilon_T, \epsilon_R), \tag{1.18}$$

$$\forall_{s,a} |Q_1(s,a) - Q_2(s,a)| \leq s(\epsilon_T, \epsilon_R), \tag{1.19}$$

where

$$s(\epsilon_T, \epsilon_R) = \frac{\epsilon_R + \gamma V_{\max} \epsilon_T}{1 - \gamma}. \tag{1.20}$$

Definition 3. *Let m be an MDP with transition function T, reward function R, state set S and action set A, and let $Q(s,a)$ and $V(s)$ be the Q-function and value function for m, respectively.*

Definition 4. *Let C be a set of* classes *for state-action pairs, and let $c : S \times A \to C$ be a* classifier*. A simple and useful example of a* classifier *would be one that assigns each state-action pair to its own* class*. For a BAMDP, where each BAMDP state is the pairing of a concrete state from the MDP and a history of all transitions observed, the classifier could assign each concrete state and action pair a* class*, ignoring history.*

Definition 5. *Let B be a knownness threshold. The quantity B needs to be chosen such that if an agent has tried some state-action pair at least B times, it has an approximately accurate estimate of its transition and reward functions.*

Definition 6. *Let $K_t \subseteq C$ be the set of* known classes, *and let $U_t = C - K_t$ be the set of* unknown classes, *at time-step t. A state-action pair (s, a) is said to be* known *if $c(s, a) \in K_t$.*

Definition 7. *Let $\mathbf{A}$ be an agent, with*

- *T_t being $\mathbf{A}$'s estimate of the transition function at time-step t,*
- *R_t being $\mathbf{A}$'s estimate of the reward function at time-step t,*
- *Q_t being $\mathbf{A}$'s estimate of the Q-function at time-step t,*
- *π_t being $\mathbf{A}$'s policy at time-step t,*

such that

$$\begin{aligned} \pi_t(s_t) &= \operatorname*{argmax}_a Q_t(s_t, a), && (1.21) \\ \forall_{(s,a)|c(s,a)\in K_t} Q_t(s, a) &= R_t(s, a) + \gamma \sum_{s'} T_t(s'|s, a) V_t(s'), && (1.22) \\ V_t(s) &= Q_t(s, \pi_t(s)). && (1.23) \end{aligned}$$

Note that $Q_t(s, a)$ is defined only for known *state-action pairs. For* unknown *state-action pairs, $\mathbf{A}$ may calculate $Q_t(s, a)$ however it wishes.*

PAC-MDP Theorem. Let the following conditions be true.

1. Optimism: for any given $c(s, a) \in U_t$, $Q_t(s, a) \geq Q(s, a) - \epsilon_u$.
2. Bounded discoveries: if $c(s_t, a_t) \in K_t$, then $\pi_{t+1} = \pi_t$,

   and $\forall_{s,a} \sum_{t=0}^{\infty} \mathbb{1}\left[(s, a) = (s_t, a_t) \wedge c(s_t, a_t) \in U_t\right] < B$.

3. Accuracy: if $c(s,a) \in K_t$, then $\forall_{s'} |T_t(s'|s,a) - T(s'|s,a)| \le \epsilon_T$ and $|R_t(s,a) - R(s,a)| \le \epsilon_R$.

Then, the expected number of sub-ϵ-optimal actions chosen by agent **A** is at most $N = \frac{1}{\delta_l} HBC$, where

$$\delta_l = \frac{\epsilon - (\epsilon_u + s(\epsilon_T, \epsilon_R) + 2\epsilon_H)}{V_{\max}}, \tag{1.24}$$

$$s(\epsilon_T, \epsilon_R) = \frac{\epsilon_R + \gamma V_{\max} \epsilon_T}{1 - \gamma}, \tag{1.25}$$

$$H = \log_\gamma \left(\frac{\epsilon_H}{V_{\max}} \right), \tag{1.26}$$

$$N \in \text{poly}\left(\frac{1}{\epsilon}, V_{\max}, \frac{1}{1-\gamma}, \log\left(\frac{\epsilon_H}{V_{\max}} \right), B, C \right). \tag{1.27}$$

Proof. Lemma 1 shows that if a certain event is sufficiently unlikely during a particular step, that step is ϵ-optimal. Lemma 2 shows that the expected number of steps without that quality is bounded by $\frac{1}{\delta\, l} HBC$. □

Definition 8. *Let D_t be the event that* **A** *visits an* unknown *state-action pair some time between time-step t and $t + H$. Or, the event $\exists_{0 \le n \le H} c(s_{t+n}, a_{t+n}) \in U_t$.*

Lemma 1. *If $P(D_t) \le \delta_l$, then $Q(s_t, a_t) \ge V(s_t) - \epsilon$. In other words, if the likelihood of a discovery within H steps is too small, a_t is an ϵ-optimal action in state s_t.*

Proof. The strategy for this proof is as follows:

- Speculate the existence of an intermediate MDP m_t, which has perfectly accurate dynamics for *known* state-action pairs, and optimistic Q-values for *unknown* state-action pairs.
- Bound the difference between the value of the current policy on the real MDP and on m_t.
- Bound the difference between the value of the current policy on m_t and **A**'s estimate.

Let m_t be an MDP with the transition function for any *known* state-action pair (with $c(s,a) \in K_t$) defined to be

$$T_{m_t}(s'|s,a) = T(s'|s,a), \tag{1.28}$$

and the Q-function for any *unknown* state-action pair (with $c(s,a) \in U_t$) defined to be

$$Q_{m_t}(s,a) = Q_t(s,a). \tag{1.29}$$

That is, for *known* state-action pairs, it has the dynamics from the true MDP, and for *unknown* state-action pairs, it uses **A**'s estimate of the Q-function.

By applying Condition 3 and the simulation lemma, we can bound the difference between its value function and **A**'s estimate:

$$|Q_{m_t}(s,a) - Q_t(s,a)| \leq s(\epsilon_T, \epsilon_R), \tag{1.30}$$

$$|V_{m_t}(s) - V_t(s)| \leq s(\epsilon_T, \epsilon_R), \tag{1.31}$$

where $s(\epsilon_T, \epsilon_R) = \frac{\epsilon_R + \gamma V_{\max}\epsilon_T}{1-\gamma}$.

The value of following policy π_t from s_t for H steps in m, or $V^{\pi_t}(s_t, H)$, versus the same value in m_t, or $V^{\pi_t}_{m_t}(s_t, H)$, can only differ if an *unknown* state-action pair is visited, and then the difference is bounded by $V_{\max}$. Therefore,

$$V^{\pi_t}(s_t, H) \geq V^{\pi_t}_{m_t}(s_t, H) - \delta_l V_{\max}, \tag{1.32}$$

$$V^{\pi_t}(s_t) + \epsilon_H \geq V^{\pi_t}_{m_t}(s_t) - \epsilon_H - \delta_l V_{\max}, \tag{1.33}$$

$$V^{\pi_t}(s_t) \geq V^{\pi_t}_{m_t}(s_t) - 2\epsilon_H - \delta_l V_{\max}, \tag{1.34}$$

$$\geq V_t(s_t) - s(\epsilon_T, \epsilon_R) - 2\epsilon_H - \delta_l V_{\max}, \tag{1.35}$$

$$\geq V(s_t) - \epsilon_u - s(\epsilon_T, \epsilon_R) - 2\epsilon_H - \delta_l V_{\max}, \tag{1.36}$$

$$Q^{\pi_t}(s_t, a_t) \geq V(s_t) - \epsilon_u - s(\epsilon_T, \epsilon_R) - 2\epsilon_H - \delta_l V_{\max}, \tag{1.37}$$

$$Q(s_t, a_t) \geq V(s_t) - \epsilon_u - s(\epsilon_T, \epsilon_R) - 2\epsilon_H - \delta_l V_{\max}. \tag{1.38}$$

Equation 1.33 is a result of applying the Bounded Horizon Lemma. Equation 1.35 is a result of applying the Simulation Lemma. Equation 1.36 is a result of applying the Optimism Condition.

If we choose δ_l such that

$$\delta_l = \frac{\epsilon - (\epsilon_u + s(\epsilon_T, \epsilon_R) + 2\epsilon_H)}{V_{\max}}, \tag{1.39}$$

then

$$Q(s_t, a_t) \geq V(s_t) - \epsilon. \tag{1.40}$$

□

Lemma 2. *The expected number of steps that* **A** *takes where* $P(D_t) > \delta_l$ *is no more than* $\frac{1}{\delta_l}HBC$*.*

Proof. Let a *discovery* be the event where $c(s_t, a_t) \in U_t$. That is, **A** took a step that may change its estimated transition and Q-functions.

The number of discoveries is at most BC, since each state-action pair's *class* can be visited at most B times before it becomes *known*, and there are C *classes* to choose from.

Let a window of H time-steps beginning at time-step t, be called a discovery window if $P(D_t) > \delta_l$, and $t = 0$, $P(L_{t-1}) \leq \delta_l$, or $c(s_{t-1}, a_{t-1}) \in U_{t-1}$. That is, these windows do not overlap. The probability of a discovery, or visiting an *unknown* state-action pair, is at least δ_l, so the expected number of discoveries per window is at least δ_l.

The expected number of windows per discovery is at most $1/\delta_l$, and since there are H steps per window, the expected number of steps per discovery is H/δ_l.

Since there are at most BC discoveries, the expected number of steps during discovery windows is $\frac{1}{\delta_l}HBC$.

Since any steps taken while not in a discovery window must must be ϵ-optimal, the expected number of sub-optimal steps is at most $\frac{1}{\delta_l}HBC$. $\square$

This proof framework provides a straightforward way to prove that certain classes of reinforcement-learning algorithms have a PAC-MDP guarantee. That is not to say that if an algorithm does not satisfy this framework's conditions, it cannot be PAC-MDP—they are sufficient, but not necessary, conditions.

1.2 Bayesian Inference

Bayesian inference refers to the general technique of combining a prior distribution with observed data to obtain a posterior distribution. Bayesian priors provide principled ways to bring knowledge into a learning problem. The prior distribution encodes the designer's assumption about how the agent's world or environment operates.

Bayesian inference is based on one simple equation, Bayes rule, that relates the prior distribution to the posterior:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}, \tag{1.41}$$

where H is the hypothesis and E is the evidence. The quantity $P(H|E)$, or the probability of the hypothesis conditioned on the evidence, is the posterior; $P(E|H)$, or the probability of the evidence conditioned on the hypothesis, is the data likelihood; $P(H)$, or the probability of the hypothesis without regard to the evidence, is the prior; $P(E)$, or the probability of the evidence without regard to the hypothesis, plays the role of a normalizing factor and is not calculated directly.

1.2.1 Coin Flipping

An example that can illustrate the Bayesian inference process is that of learning the bias of a coin by flipping it a number of times.

Suppose there is a bag of 10000 coins, 10 of which are two-headed. The remaining 9990 coins are fair coins, equally likely to flip heads or tails. The experiment is to take a coin from this bag, and track how likely it is that the coin is biased after some number of n observed flips.

First, there must be a generative model to describe the experiment dynamics:

$$\begin{aligned} \rho &\sim \begin{cases} 0.5 & \text{w.p.} \quad 9990/10000, \\ 1 & \text{w.p.} \quad 10/10000, \end{cases} && (1.42) \\ H &\sim \text{Binomial}(\rho, n), && (1.43) \end{aligned}$$

where ρ is the bias of the coin, either 0.5 (fair) or 1 (two-headed), n is the number of flips made, and H is the number of flips observed to be heads.

The number of observed heads, H, is drawn from the binomial distribution, which has the probability mass function

$$P(H = h|\rho, n) = \binom{n}{h} h^{\rho}(n-h)^{1-\rho}. \tag{1.44}$$

The prior distribution of the coin's bias is defined explicitly in the generative model, in Equation 1.42. To find the corresponding posterior distribution, one must use Bayes rule from Equation 1.41.

The posterior, $P(\rho|h, n)$, can be formulated by applying Bayes rule.

$$\begin{aligned} P(\rho|h,n) &\propto P(h|\rho,n)P(\rho|n), && (1.45)\\ &\propto P(h|\rho,n)P(\rho) && (1.46)\\ &\propto \binom{n}{h} h^{\rho}(n-h)^{1-\rho}P(\rho) && (1.47)\\ &\propto h^{\rho}(n-h)^{1-\rho}P(\rho). && (1.48) \end{aligned}$$

From our original model, Equation 1.42 tells us that there are only two possibilities for ρ: 0 or 0.5. To find their exact posterior likelihoods, we can evaluate Equation 1.46 for both and normalize.

$$\begin{aligned} P(\rho = 0.5|H = h, n) &\propto 0.5^h 0.5^{n-h} \cdot \frac{9990}{10000} && (1.49)\\ &\propto 0.5^n \cdot \frac{9990}{10000}. && (1.50) \end{aligned}$$

$$P(\rho = 1|H = h, n) \propto 1^h 0^{n-h} \cdot \frac{10}{10000}. \quad (1.51)$$

Let's first evaluate the posterior likelihood that $\rho = 1$.

$$P(\rho = 1|H = h, n) = \frac{1^h 0^{n-h} \cdot \frac{10}{10000}}{1^h 0^{n-h} \cdot \frac{10}{10000} + 0.5^n \cdot \frac{9990}{10000}}. \quad (1.52)$$

There are two basic cases:

1. Every flip was heads, or $h = n$.

2. Not every flip was heads, or $h < n$.

In Case 1, when $h = n$, Equation 1.52 can be used to evaluate the posterior exactly.

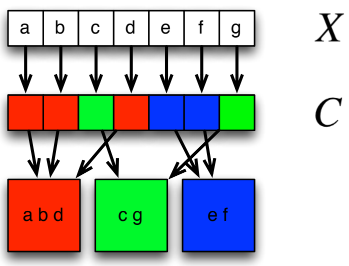

Figure 1.4

In Case 2, when $h < n$, $P(\rho = 1|H = h, n)$ goes to zero because of the 0^{n-h} term in the numerator.

This same process, applying Bayes rule to turn a prior and observations into a posterior, is the basis for all models discussed in this work.

1.2.2 The Chinese Restaurant Process

The Chinese Restaurant Process, or CRP, is a nonparametric prior that can be used to guess assignments of elements in a set to clusters [13]. Here, the word nonparametric indicates that the number of parameters supplied by the CRP posterior is not known in advance and instead depends on the size of the observation set that it is given.

The CRP prior likelihood is easy to evaluate online. For a new element, the prior likelihood that it belongs to any given cluster is proportional to the number of elements already in that cluster, and the prior likelihood that it belongs in a new cluster is proportional to a parameter α.

Inference on mixture models can often be done with a CRP. A generative mixture model follows. The observed data X and the base measure G_0 are given, and α is a parameter that influences cluster size:

$$\begin{aligned} F_i &\sim G_0 && (1.53)\\ C &\sim CRP(\alpha) && (1.54)\\ X_t &\sim F_{C_t}. && (1.55) \end{aligned}$$

Here, C is a vector and C_t is the index of the cluster assigned to the data point X_t. F_i is the sampling distribution associated with cluster i, and is drawn from the base measure G_0. Figure 1.4 illustrates how the data in X is clustered by C, the assignment from the CRP.

The task here is to perform inference on the assignments C and the cluster distributions F_i. It is useful to choose cluster distributions F and base measure G_0 such that G_0 is conjugate prior to F. Conjugacy allows us to easily derive a posterior $F \sim G_0|D$ and to calculate the "clustering likelihood" $P(X|C, G_0) = \prod_i \int_{F_i} F_i(X^i)G_0(F_i)dF_i$ analytically, where $X^i = \{X_t|C_t = i\}$ is the collection of data from X assigned to cluster i.

The conjugacy between F_i and G_0 allows us to sample the assignment vector C with no knowledge of the parameters for F_i. C is sampled from the distribution $P(C|X, G_0, \alpha) \propto P(X|C, G_0)P(C|\alpha)$. This step in itself is useful, as clustering has been performed. If more data needs to be generated for simulation, a new point X_j's cluster can be chosen according to $P(C_j|C_{-j}, \alpha)$, that cluster's distribution F_{C_j} can be chosen from $G_0(F_{C_j}|X^{C_j})$ and X_j's value can be chosen according to $F_{C_j}(X_j)$.

There is also a closed form for the CRP prior, $P(C|\alpha)$:

$$P(C|\alpha) = \alpha^r \frac{\Gamma(\alpha)}{\Gamma(\alpha + \sum_i n_i)} \prod_i \Gamma(n_i), \quad (1.56)$$

where $n_i = \sum_j \delta_{C_j,i}$, and $\delta_{i,j} = 1$ iff $i = j$, is the number of elements assigned to cluster i, and r is the number of clusters with at least 1 member.

The CRP is exchangeable. That is, the order of the elements does not affect the posterior assignment distribution.

It is worth noting that the number of possible vectors C grows exponentially with the number of elements in X. For CRP inference, Gibbs sampling [14, 15] is a practical way to do approximation.

1.3 Bayesian Models for Reinforcement Learning

Inferring latent structure from observations is a difficult task in any context, let alone in reinforcement learning. Building structure into a learning algorithm via an expert runs the risk of assuming too much and overfitting, or too little and not being useful when compared to an unstructured learner. By the nature of learning in stochastic systems (such as an environment used for reinforcement learning), creating a system that infers structure from the observations made can only effectively be done by weighing the likelihood that the observations came from a particular model against the likelihood that the model was used in the first place, and this is exactly the Bayesian approach to model-based reinforcement learning. While a frequentist approach might consider only the observation likelihood, weighing that likelihood against a prior probability allows the algorithm to filter out unlikely models early on, often with very little data. Instead of saying, "I do not know, and that is all I know," before the necessary data is collected, a Bayesian model can say "I am not certain, but I am leaning towards a few possibilities." Since, when learning structure, it is important to be able to consider different possible models, the ability to narrow down the options quickly is key to making a reinforcement-learning algorithm that finds an optimal policy quickly. Without the Bayesian approach to learning, it is very difficult to generate such flexible, adaptive behavior.

As mentioned briefly at the beginning of this chapter, the Bayesian approach to learning can be applied to modeling reinforcement-learning domains. There exist Bayesian approaches to model-free reinforcement-learning [16], but they do not fit within the scope of this dissertation devoted to model-based reinforcement-learning. In particular, they are a step removed from the environment structure, since they are priors on the value function or on the policy. They do not provide a sensible way for a system designer to provide prior knowledge about environment structure.

For model building for reinforcement learning, we must first start with a model prior. We combine it with observations made by the agent to derive a model posterior. This model posterior can be used for reasoning about future choices that the agent will make.

1.3.1 Flat Dirichlet Multinomial

For a discrete-state -action MDP, the Flat Dirichlet Multinomial, or FDM, prior can be used [17]. According to the FDM prior, the next-state distributions for all state-action pairs are multinomials whose parameters are drawn i.i.d. from the Dirichlet distribution, parameterized by some constant vector α:

$$\theta^{s,a} \sim \text{Dirichlet}(\alpha), \tag{1.57}$$

$$N^{s,a} \sim \text{Multinomial}(\theta^{s,a}). \tag{1.58}$$

Because the Dirichlet distribution is conjugate to the multinomial distribution, the FDM posterior is easy to compute:

$$\theta^{s,a} | N^{s,a} \sim \text{Dirichlet}(\alpha + N^{s,a}), \tag{1.59}$$

$$P(s' | N^{s,a}, \alpha) \propto \alpha_{s'} + N^{s,a}_{s'}. \tag{1.60}$$

The derivation for Equations 1.59 and 1.60 are a result of Equation 2.12 in Chapter 2.

Equation 1.59 gives us the ability to sample from the MDP posterior. Equation 1.60 gives us the ability to sample experience, and as a result FDM can be considered a trajectory prior. This capability is useful if the planner wants only a simulation oracle, and doesn't care about entire models.

This prior is theoretically attractive, since it easily supports all possible discrete-state discrete-action MDPs. but not good at generalizing as it treats estimating each state-action as a separate learning problem. On the other hand, a different choice of the α parameter for each state-action pair can be used to essentially *know* the MDP before hand - with some uncertainty built in where appropriate. RL agents that rely on the FDM prior generally are unlikely outperform non-Bayesian agents; if the prior knowledge states, essentially, that it is an MDP and the transition functions can be any multinomial, this information does not give the Bayesian agent a leg up on other agents, since this assumption will usually be baked into the algorithm design from the beginning.

1.3.2 Gaussian Bandits

For a bandit problem in which the reward for an arm can be any real number, the Gaussian bandit prior can be used [18]:

$$\mu^a \sim N(\mu_0, \sigma_0^2), \tag{1.61}$$

$$R_t^a \sim N(\mu^a, \sigma^2). \tag{1.62}$$

In Equations 1.61 and 1.62, μ_0 and σ_0 are parameters indicating where the reward averages are likely to be, and σ is a parameter indicating the variance in the reward seen for a particular arm. The value μ^a is the expected reward acquired from pulling arm a, and R^a_t is the reward acquired from pulling arm a for the t^{th} time.

Like the FDM prior in Section 1.3.1, the Gaussian bandit prior makes use of conjugacy for efficient inference. The known-variance Normal distribution is its own conjugate prior:

$$\mu^a|R^a \sim N\left(\left(\frac{\mu_0}{\sigma_0^2}+\frac{\sum_{t=1}^{T}R^a_t}{\sigma^2}\right)/\left(\frac{1}{\sigma_0^2}+\frac{T}{\sigma^2}\right),\left(\frac{1}{\sigma_0^2}+\frac{T}{\sigma^2}\right)^{-1}\right). \quad (1.63)$$

While the Gaussian bandit prior does not model all possible continuous-reward distributions like the FDM prior models all possible discrete MDPs, it can still be used to trade-off uncertainty in the mean (indicated by the σ_0^2 parameter) and knowledge of the mean gathered from data, expressed in Equation 1.63.

1.4 Model-based Bayesian Reinforcement Learning

This dissertation focuses on developing models appropriate to reinforcement-learning domains, the algorithms that can take advantage of these models to effectively solve the reinforcement-learning problem, and the analysis of these algorithms.

Chapter 2 will extend the ideas from Section 1.2 and Section 1.3 to create more structured and useful models.

Chapter 3 is a survey of existing model-based and Bayesian model-based reinforcement-learning algorithms.

Chapter 4 introduces the **Best Of Sampled Set** algorithm and its analysis.

Chapter 5 introduces the **Bayesian Forward Search Sparse Sampling** algorithm and its analysis.

Chapter 6 reports on several experimental results comparing Bayesian model-based reinforcement-learning algorithms against each other and other model-based reinforcement-learning algorithms.

Chapter 7 concludes the dissertation and summarizes its contributions.

Chapter 2
Models and Inference

A model, when used as a prior for a reinforcement-learning agent, is a way to inform the agent about structure that it can expect to see in the environment. Non-Bayesian approaches to structured learning in RL, such as **RAM-RMAX** [5], are often much more rigid in their structure. Even when the rigidity is relaxed [19], the **RMAX**-based algorithm is extremely conservative, and must try sub-optimal actions many times before it is satisfied that the MDP is learned sufficiently well. The Bayesian approach to learning structure, in conjunction with an appropriate model-based Bayesian reinforcement learning algorithm, allows an agent to focus on models that are *likely*, rather than ruling out all models that are still *possible*. Additionally, priors are inherently composable, as is demonstrated in Section 6.3.4, giving the algorithm the ability to consider multiple models that may or may not share structural elements. The end result is much more intelligent behavior, balancing the need to obtain high reward with the drive to learn more about the environment.

In this chapter, I will discuss several different priors that can be used for model inference in a reinforcement–learning setting. The non-parametric models in Section 2.3, as well as the environment-specific models in Chapter 6, are novel work. The analysis in Section 2.4, on the sample complexity of models and priors, is a contribution of this dissertation. Before this work, most Bayesian approaches to reinforcement learning focused on the simple and easy-to-work with Flat Dirichlet Multinomial, which is detailed in Section 2.1. There are exceptions that will be discussed in Chapter 3 [20, 21, 22], and the models discussed in this chapter can work equally well with any of those algorithms.

There are two basic types of distributions that will be discussed: MDP distributions and experience distributions. The result of sampling from an MDP distribution is an entire MDP which can be used with various Bayesian algorithms to attack the reinforcement–learning problem. An experience distribution is one whose posterior is a theoretical transition in the agent's world.

An experience distribution Θ is a distribution over possible next state/reward pairs s', r, given a possible history h, state s, and action a:

$$s', r \quad \sim \quad \Theta | s, a, h. \tag{2.1}$$

It is important to note that neither s nor h need be the agent's current state or history. They are possible states and histories that can be used for internal simulation.

An MDP distribution ϕ is a distribution over possible MDPs, given a history h:

$$m \quad \sim \quad \phi | h. \tag{2.2}$$

Like the history for experience distributions in Equation 2.1, the history in Equation 2.2 may or may not be the agent's current history (though often it will be).

It is always possible to use an MDP distribution to mimic an experience distribution by first sampling a model, and then sampling a single step using that model:

$$\begin{aligned} m &\sim \phi|h, && (2.3) \\ s' &\sim T_m(s,a), && (2.4) \end{aligned}$$

where $T_m(s,a)$ is the MDP m's next-state distribution when in state s and taking action a.

All of the model-based Bayesian reinforcement learning algorithms discussed in this work use one of these two basic types of priors.

2.1 Dirichlet Models

One of the simplest Bayesian models for reinforcement learning is the Flat Dirichlet Multinomial, or FDM. The FDM is a prior for discrete state and action spaces with independent next-state distributions for each state-action pair. That is, next-state observations from one particular state and action gives no information about the next-state distribution for some other state and action.

The "flat" part of FDM refers to the fact that all pieces of the model must be learned independently; no higher level information can be shared. The "multinomial" part refers to the fact that each next-state distribution can be described by a multinomial. The "Dirichlet" part refers to the fact that the prior for each multinomial is the Dirichlet distribution.

The Dirichlet distribution is defined as follows:

$$\mathrm{Dir}(\theta|\alpha) = \frac{\Gamma\left(\sum_i \alpha_i\right)}{\prod_i \Gamma(\alpha_i)} \prod_i \theta_i^{\alpha_i - 1}. \tag{2.5}$$

According to the FDM prior, each state-action pair's next-state distribution is a multinomial drawn i.i.d. from the Dirichlet distribution with parameter α, a vector of non-negative numbers that represent an initial indication of possible next-state distributions. Each entry corresponds to an entry in the multinomial, which indicates the likelihood of a particular next-state. The α-vector is a parameter to the model, and can be tuned to particular domains.

In general, an α of all 1s means that any possible multinomial is equally likely. An α with very high entries indicates that the multinomial is essentially known, and has weights corresponding to the ratios of the elements in α.

Due to a conjugacy relationship between the multinomial and Dirichlet distributions, we can easily derive a posterior for θ given α and the observations X.

$$\begin{aligned}
P(\theta|X,\alpha) &\propto P(X|\theta,\alpha)P(\theta|\alpha), && (2.6)\\
&\propto \text{Mult}(X|\theta)\text{Dir}(\theta|\alpha), && (2.7)\\
&\propto \frac{(\sum_i X_i)!}{\prod_i X_i!}\prod_i \theta_i^{X_i}\frac{\Gamma\left(\sum_i \alpha_i\right)}{\prod_i \Gamma(\alpha_i)}\prod_i \theta_i^{\alpha_i-1}, && (2.8)\\
&\propto \prod_i \theta_i^{X_i}\prod_i \theta_i^{\alpha_i-1}, && (2.9)\\
&\propto \prod_i \theta_i^{X_i+\alpha_i-1}, && (2.10)\\
&\propto \frac{\Gamma\left(\sum_i X_i+\alpha_i\right)}{\prod_i \Gamma(X_i\alpha_i)}\prod_i \theta_i^{X_i+\alpha_i-1}, && (2.11)\\
&= \text{Dir}(\theta|\alpha+X). && (2.12)
\end{aligned}$$

The α vector can encode what is learned from a set of prior observations. With a $\text{Dir}(\langle x,y,z\rangle)$ prior and an observation of the second element, the posterior is $\text{Dir}(\langle x,y+1,z\rangle)$. Moving forward with that posterior is equivalent to starting with the prior $\text{Dir}(\langle x,y+1,z\rangle)$ without the benefit of the observation. It follows that starting with a prior of $\text{Dir}(\langle 1+x,1+y,1+z\rangle)$ is the same as starting with a prior of $\text{Dir}(\langle 1,1,1\rangle)$ and making x, y, and z observations of the first, second, and third outcomes, respectively.

The likelihood of any particular observation set, given the parameter α, can be derived from the model

$$\begin{aligned} \theta &\sim \text{Dir}(\alpha), &(2.13)\\ X &\sim \text{Mult}(\theta, n), &(2.14)\end{aligned}$$

where X is the result of n die rolls using θ as the side likelihoods, we can derive the following by integrating out θ.

$$\begin{aligned}
P(X|\alpha, n) &= \int_\theta \text{Mult}(X|\theta, n)\text{Dir}(\theta|\alpha), &(2.15)\\
&= \int_\theta \frac{(\sum_i X_i)!}{\prod_i X_i!}\prod_i \theta_i^{X_i}\frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)}\prod_i \theta_i^{\alpha_i - 1}, &(2.16)\\
&= \frac{(\sum_i X_i)!}{\prod_i X_i!}\frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)}\int_\theta \prod_i \theta_i^{X_i+\alpha_i-1}, &(2.17)\\
&= \frac{(\sum_i X_i)!}{\prod_i X_i!}\frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)}\frac{\prod_i \Gamma(\alpha_i + X_i)}{\Gamma(\sum_i \alpha_i + X_i)}\int_\theta \frac{\Gamma(\sum_i \alpha_i + X_i)}{\prod_i \Gamma(\alpha_i + X_i)}\prod_i \theta_i^{X_i+\alpha_i-1}, &(2.18)\\
&= \frac{(\sum_i X_i)!}{\prod_i X_i!}\frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)}\frac{\prod_i \Gamma(\alpha_i + X_i)}{\Gamma(\sum_i \alpha_i + X_i)}\int_\theta \text{Dir}(\theta|\alpha + X), &(2.19)\\
&= \frac{(\sum_i X_i)!}{\prod_i X_i!}\frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)}\frac{\prod_i \Gamma(\alpha_i + X_i)}{\Gamma(\sum_i \alpha_i + X_i)}. &(2.20)
\end{aligned}$$

Moving from Equation 2.19 to Equation 2.20 is possible because the integral of any distribution over its entire support necessarily sums to 1.

2.1.1 Tied Dirichlet Models

By itself, the FDM prior does not allow any sharing of information between states. The observations made of outcomes from any given state-action pair cannot be used to make inferences about any other state-action pair. However, sometimes the algorithm designer has prior knowledge indicating that the *outcome* distributions of two or more state-action pairs are identical. Here, we use the term *outcome* to be distinct from *next-state*: the outcome must be combined with the starting state to get the next-state. For instance, the *move-left* outcome can have the same likelihood for two particular states, but the identity of the actual state to the left differs.

If these assumptions are made, we can share the outcome multinomial for several state-action pairs. Then, outcome observations made in one of the state-action pairs can be used to make inferences about the outcome distribution for any of the other state-action pairs in the set in question.

For instance, in the *Marble Maze* domain [5], the possible outcomes are *north*, *east*, *south*, *west*, and *stay*. They each indicate the agent moving to the corresponding adjacent cell in the maze (or not moving). No other outcomes are possible: the agent cannot move to non-adjacent cells. The outcome distribution for any state-action pair depends on the walls surrounding the cell for the current state. If there is a wall to the north, then any outcome that would have been *north* becomes *stay*, indicating that the agent ran into the wall and was unable to move.

Since the outcome distribution is a function of the current cell's walls, different states whose cell wall configurations are identical have identical outcome distributions. If these wall configurations are known before the experiment begins, then a tied Dirichlet model can be used to speed learning.

$$\theta \sim \mathrm{Dir}(\alpha), \tag{2.21}$$

$$X^j \sim \mathrm{Mult}(\theta), \tag{2.22}$$

$$Y = \langle X^1, X^2, ..., X^J \rangle. \tag{2.23}$$

For a vector V, let $\frac{(\sum_i V_i)!}{\prod_i V_i!} = \frac{(\sum_i V_i)!}{\prod_i V_i!}$, and $\frac{\Gamma(\sum_i V_i)}{\prod_i \Gamma(V_i)} = \frac{\Gamma(\sum_i V_i)}{\prod_i \Gamma(V_i)}$. For vectors V and ρ, let $\rho^V = \prod_i \rho_i^{V_i}$. For a vector V and a constant k, let $V + k = \langle V_1 + k, V_2 +$

$k, \ldots \rangle$.

$$\begin{aligned}
P(Y|\alpha) &= \int_\theta P(Y|\theta)P(\theta|\alpha), && (2.24)\\
&= \int_\theta \prod_j \mathrm{Mult}(X^j|\theta)\mathrm{Dir}(\theta|\alpha), && (2.25)\\
&= \int_\theta \prod_j \frac{\left(\sum_i X_i^j\right)!}{\prod_i X_i^j!}\theta^{X^j}\frac{\Gamma\left(\sum_i \alpha_i\right)}{\prod_i \Gamma(\alpha_i)}\theta^{\alpha-1}, && (2.26)\\
&= \prod_j \frac{\left(\sum_i X_i^j\right)!}{\prod_i X_i^j!}\frac{\Gamma\left(\sum_i \alpha_i\right)}{\prod_i \Gamma(\alpha_i)}\int_\theta \prod_j \theta^{X^j+\alpha-1}, && (2.27)\\
&= \prod_j \frac{\left(\sum_i X_i^j\right)!}{\prod_i X_i^j!}\frac{\Gamma\left(\sum_i \alpha_i\right)}{\prod_i \Gamma(\alpha_i)}\int_\theta \prod_j \frac{\Gamma\left(\sum_i X^j+\alpha_i\right)}{\prod_i \Gamma(X^j+\alpha_i)}\frac{\prod_i \Gamma(X^j+\alpha_i)}{\Gamma\left(\sum_i X^j+\alpha_i\right)}\theta^{X^j+\alpha-1} && (2.28)\\
&= \prod_j \frac{\left(\sum_i X_i^j\right)!}{\prod_i X_i^j!}\frac{\Gamma\left(\sum_i \alpha_i\right)}{\prod_i \Gamma(\alpha_i)}\frac{\prod_i \Gamma(X^j+\alpha_i)}{\Gamma\left(\sum_i X^j+\alpha_i\right)}\int_\theta \mathrm{Dir}(\theta|\alpha+X^j), && (2.29)\\
&= \prod_j \frac{\left(\sum_i X_i^j\right)!}{\prod_i X_i^j!}\frac{\Gamma\left(\sum_i \alpha_i\right)}{\prod_i \Gamma(\alpha_i)}\frac{\prod_i \Gamma(X^j+\alpha_i)}{\Gamma\left(\sum_i X^j+\alpha_i\right)}. && (2.30)
\end{aligned}$$

Section 2.3 demonstrates how to automatically tie different states together based on experience using nonparametric clustering techniques.

2.2 Gaussian Models

Similarly to the Dirichlet model's easy relation with discrete state and action spaces, a Gaussian model can be used to provide priors for continuous spaces. If an environment's dynamics can be modeled with a Gaussian next-state or outcome function, there are convenient priors that can be used to make inferences.

For discrete action domains, learning the dynamics for each action can be considered a separate problem. Section 2.3 discusses methods to tie information from different actions and different states together, but for now we'll study the simpler learning problem where each action can be considered in isolation.

The simplest case is when the next-state is drawn directly from a Gaussian distribution. A more interesting case has an outcome drawn from a Gaussian distribution, and then the next-state is derived from applying the outcome to the previous state. The inference problem, where the agent learns the mean and variance of the Gaussian distribution in question, is the same either way. The only difference is what data the inference engine is given as examples.

An easy version of this problem is the unknown mean, known variance prior: given some covariance Σ, and a mean prior of $N(\mu_\mu, \Sigma_\mu)$, we can concisely describe the model as follows:

$$\mu_a \sim N(\mu_\mu, \Sigma_\mu), \tag{2.31}$$

$$o_t \sim N(\mu_{a_t}, \Sigma), \tag{2.32}$$

$$s_{t+1} = f(s_t, o_t). \tag{2.33}$$

Here, μ_μ is the prior mean for the latent variable μ_a, and Σ_μ is the prior covariance for μ_a. a_t is the action performed by the agent on the t^{th} timestep, and o_t is the outcome observed as a result of that action. The next state, s_{t+1} can be derived by applying the outcome function f to the previous state s_t and outcome.

Given this model and a history $s_0, a_0, o_0, ..., s_T, a_T, o_T$, a posterior distribution for latent model variables μ_a can be inferred, for all actions $a \in A$. Since the Gaussian distribution is conjugate prior to the known-covariance multivariate normal distribution, there is a closed-form solution for its posterior [23]:

$$\Sigma' = \left(\Sigma_u^{-1} + n\Sigma^{-1}\right)^{-1}, \tag{2.34}$$

$$\bar{x} = \frac{1}{T}\sum_{t=1}^{T} o_t, \tag{2.35}$$

$$\mu_a \sim N\left((\Sigma'\left(\Sigma_u^{-1}\mu_u + T\Sigma^{-1}\bar{x}\right), \Sigma'\right). \tag{2.36}$$

It is possible to do inference with a known mean, unknown variance prior, described as follows:

$$\Sigma_a \sim W^{-1}(m, \Psi), \tag{2.37}$$

$$o_t \sim N(\mu, \Sigma_{a_t}), \tag{2.38}$$

$$s_{t+1} = f(s_t, o_t). \tag{2.39}$$

Here, $W^{-1}(m, \Psi)$ is the inverse Wishart distribution, whose density is described in Equation 2.40:

$$W^{-1}(\Sigma|m, \Psi) = \frac{|\Psi|^{\frac{m}{2}}}{2^{mp/2}\Gamma_p(m/2)}|\Sigma|^{-\frac{m+p+1}{2}} \exp\left(-\frac{1}{2}\text{tr}\left[\Psi\Sigma^{-1}\right]\right), \tag{2.40}$$

where p is the number of dimensions, and Γ_p is the partial gamma function.

The inverse Wishart distribution is chosen because it is conjugate (and therefore mathematically convenient) to the known mean, unknown variance Gaussian, and as a result inference on the covariance Σ is efficient. Let $X = \begin{bmatrix} o_1 & o_2 & \dots & o_n \end{bmatrix}$ be a matrix whose columns are the observed outcome vectors. Assuming, without loss of generality, that the mean is zero, the covariance posterior is defined as follows:

$$
\begin{aligned}
& P(\Sigma|X,\mu,m,\Psi) && (2.41)\\
\propto\ & P(X|\Sigma,\mu,m,\Psi)P(\Sigma|\mu,m,\Psi), && (2.42)\\
\propto\ & N(X|\mu,\Sigma)W^{-1}(\Sigma|m,\Psi), && (2.43)\\
\propto\ & \begin{array}{l}(2\pi)^{-\frac{np}{2}}|\Sigma|^{-\frac{n}{2}}\exp\left(-\frac{1}{2}tr\left[XX'\Sigma^{-1}\right]\right)\\ \cdot\ \frac{|\Psi|^{\frac{m}{2}}}{2^{\frac{mp}{2}}\Gamma_p(\frac{m}{2})}|\Sigma|^{-\frac{m+p+1}{2}}\exp\left(-\frac{1}{2}tr\left[\Psi\Sigma^{-1}\right]\right),\end{array} && (2.44)\\
\propto\ & |\Sigma|^{-\frac{n}{2}}\exp\left(-\frac{1}{2}tr\left[XX'\Sigma^{-1}\right]\right)|\Sigma|^{-\frac{m+p+1}{2}}\exp\left(-\frac{1}{2}tr\left[\Psi\Sigma^{-1}\right]\right), && (2.45)\\
\propto\ & |\Sigma|^{-\frac{m+n+p+1}{2}}\exp\left(-\frac{1}{2}tr\left[XX'\Sigma^{-1}+\Psi\Sigma^{-1}\right]\right), && (2.46)\\
\propto\ & |\Sigma|^{-\frac{m+n+p+1}{2}}\exp\left(-\frac{1}{2}tr\left[(XX'+\Psi)\Sigma^{-1}\right]\right), && (2.47)\\
\propto\ & \frac{|\Psi|^{\frac{m+n}{2}}}{2^{\frac{(m+n)p}{2}}\Gamma_p(\frac{m+n}{2})}|\Sigma|^{-\frac{m+n+p+1}{2}}\exp\left(-\frac{1}{2}tr\left[(XX'+\Psi)\Sigma^{-1}\right]\right), && (2.48)\\
=\ & W^{-1}(\Sigma|m+n,\Psi+XX'). && (2.49)
\end{aligned}
$$

We can also assess the data likelihood. Offsetting $x_1, x_2, ..., x_n$ to have a mean of $\mu = 0$,

$$
\begin{aligned}
& P(X|\mu, m, \Psi) && (2.50)\\
= \;& \int_{\Sigma} P(X|\mu, \Sigma, m, \Psi) P(\Sigma|m, \Psi) d\Sigma, && (2.51)\\
= \;& \int_{\Sigma} N(X|\mu, \Sigma) W^{-1}(\Sigma|m, \Psi) d\Sigma, && (2.52)\\
= \;& \int_{\Sigma} \left[\prod_{i=1}^{n} N(x_i|\mu, \Sigma) \right] W^{-1}(\Sigma|m, \Psi) d\Sigma, && (2.53)\\
= \;& \int_{\Sigma} \left(\begin{array}{c} \left[\prod_{i=1}^{n} \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2} x_i' \Sigma^{-1} x_i) \right] \\ \cdot \frac{|\Psi|^{m/2}}{2^{mp/2}\Gamma_p(m/2)} |\Sigma|^{-\frac{m+p+1}{2}} \exp\left(-\frac{1}{2} \mathrm{tr} \left[\Psi \Sigma^{-1} \right] \right) d\Sigma \end{array} \right), && (2.54)\\
= \;& \int_{\Sigma} \left(\begin{array}{c} \left[\prod_{i=1}^{n} \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2} tr \left[XX' \Sigma^{-1} \right]) \right] \\ \cdot \frac{|\Psi|^{m/2}}{2^{mp/2}\Gamma_p(m/2)} |\Sigma|^{-\frac{m+p+1}{2}} \exp\left(-\frac{1}{2} \mathrm{tr} \left[\Psi \Sigma^{-1} \right] \right) d\Sigma \end{array} \right), && (2.55)\\
= \;& \int_{\Sigma} \left(\begin{array}{c} \frac{1}{(2\pi)^{np/2}|\Sigma|^{n/2}} \left[\exp(-\frac{1}{2} tr \left[XX' \Sigma^{-1} \right]) \right] \\ \cdot \frac{|\Psi|^{m/2}}{2^{mp/2}\Gamma_p(m/2)} |\Sigma|^{-\frac{m+p+1}{2}} \exp\left(-\frac{1}{2} \mathrm{tr} \left[\Psi \Sigma^{-1} \right] \right) d\Sigma \end{array} \right), && (2.56)\\
= \;& \begin{array}{l} \frac{1}{(2\pi)^{np/2}} \frac{|\Psi|^{m/2}}{2^{mp/2}\Gamma_p(m/2)} \\ \cdot \int_{\Sigma} \left(|\Sigma|^{-\frac{m+n+p+1}{2}} \exp(-\frac{1}{2} tr \left[XX'\Sigma^{-1} + \Psi\Sigma^{-1} \right]) d\Sigma \right), \end{array} && (2.57)\\
= \;& \begin{array}{l} \frac{1}{(2\pi)^{np/2}} \frac{|\Psi|^{m/2}}{2^{mp/2}\Gamma_p(m/2)} \\ \cdot \int_{\Sigma} \left(|\Sigma|^{-\frac{(m+n)+p+1}{2}} \exp(-\frac{1}{2} tr \left[(XX + \Psi)\Sigma^{-1} \right]) d\Sigma \right), \end{array} && (2.58)\\
= \;& \begin{array}{l} \frac{1}{(2\pi)^{np/2}} \frac{|\Psi|^{m/2}}{2^{mp/2}\Gamma_p(m/2)} \frac{2^{(m+n)p/2}\Gamma_p((m+n)/2)}{|XX'+\Psi|^{(m+n)/2}} \\ \cdot \int_{\Sigma} \left(|\Sigma|^{-\frac{(m+n)+p+1}{2}} \frac{|XX'+\Psi|^{(m+n)/2}}{2^{(m+n)p/2}\Gamma_p((m+n)/2)} \exp(-\frac{1}{2} tr \left[(XX + \Psi)\Sigma^{-1} \right]) d\Sigma \right), \end{array} && (2.59)\\
= \;& \begin{array}{l} \frac{1}{(2\pi)^{np/2}} \frac{|\Psi|^{m/2}}{2^{mp/2}\Gamma_p(m/2)} \frac{2^{(m+n)p/2}\Gamma_p((m+n)/2)}{|XX'+\Psi|^{(m+n)/2}} \\ \cdot \int_{\Sigma} W^{-1}(\Sigma|m+n, XX'+\Psi) d\Sigma, \end{array} && (2.60)\\
= \;& \pi^{-\frac{np}{2}} \frac{|\Psi|^{\frac{m}{2}}}{|XX'+\Psi|^{\frac{m+n}{2}}} \frac{\Gamma_p(\frac{m+n}{2})}{\Gamma_p(\frac{m}{2})} \int_{\Sigma} W^{-1}(\Sigma|m+n, XX'+\Psi) d\Sigma. && (2.61)
\end{aligned}
$$

Since we know that all distributions necessarily sum to 1, we can remove the integral in Equation 2.61 to see that

$$P(X|\mu, m, \Psi) = \pi^{-\frac{np}{2}} \frac{|\Psi|^{\frac{m}{2}}}{|XX' + \Psi|^{\frac{m+n}{2}}} \frac{\Gamma_p(\frac{m+n}{2})}{\Gamma_p(\frac{m}{2})}, \tag{2.62}$$

which is useful for the mixture models in Section 2.3.

2.3 Nonparametric Mixture Models

While the application of conjugacy concepts to a reinforcement-learning scenario is interesting and can be fun for the enthusiastic Bayesian, models that don't provide any flexibility or effective way to incorporate interesting prior knowledge have limited utility. In most reinforcement-learning problems, observations made in one state can help you make inferences about other similar states with less data.

If the algorithm designer's prior knowledge says that there are groups of states that share outcome dynamics, it makes sense to cluster states together based on similarities seen in their next-state distributions. Many classical machine-learning techniques that accomplish this task must be provided with a guess for the total number of clusters.

In this section, we will discuss the nonparametric[1] Bayesian approach to clustering.

2.3.1 Dirichlet Mixture Models

With discrete state- and action-space environments whose states are divided into several (unknown) groups based on their dynamics, a Dirichlet mixture model can be used to help inference. Here, s_1 and s_2 refer to two arbitrary (yet

[1] Here *nonparametric* refers to the fact that the model can have an arbitrarily large number of parameters.

different) states in the MDP, rather than the states observed on the first and second steps in an experiment. Let $o_{s,a}$ be a vector of counts, representing a histogram of the outcomes observed so far in state s when taking action a.

The clustering model follows:

$$\begin{aligned} C &\sim CRP(\alpha), && (2.63)\\ \theta_{z,a} &\sim Dirichlet(\beta), && (2.64)\\ o_{s,a} &\sim Mult(\theta_{C_s,a}). && (2.65) \end{aligned}$$

In this model, C is the set of assignments of states to clusters. So, C_1 is the number representing the cluster to which s_1 belongs. The quantity $\theta_{z,a}$ is the parameter to the dynamics for states in cluster z when performing action a. Finally, the outcome histogram $o_{s,a}$ is drawn from a multinomial parameterized by the θ corresponding to the cluster containing the state in question.

Using this model, we can sample dynamics from the posterior $P(\theta, C|o)$ to get a guess about the underlying MDP. Because the Dirichlet distribution is conjugate to the multinomial distribution, the likelihood $P(o|C)$ can be evaluated efficiently. Using that likelihood, approximation techniques, such as Gibbs sampling, can be used to effectively sample the posterior $P(C|o) \propto P(o|C)P(C)$, and then sampling from $P(\theta|C, o)$ is straightforward.

The posterior sampling process, using Gibbs sampling, is as follows. Start with a guess for the value of C. Then, choose one state i to move to different cluster. For all possible clusters j it could belong to, including a new one, find the likelihood $P(C^{i,j}|o)$, where $C^{i,j}$ is the same as C except that state i is now in cluster j. Assign i to cluster j with probability proportional to $P(C^{i,j}|o)$. Then, move to the next state. Continue sweeping through all the states, reassigning them to different clusters, until the distribution is suitably mixed [2] Once C is chosen, then each cluster's multionimial is sampled from a Dirichlet whose posterior uses the evidence from the states in that cluster.

2.3.2 Gaussian Mixture Models

The use of nonparametric clustering techniques can also be applied to Gaussian dynamics. Relocatable Outcomes Across Regions, or ROAR is the name I have given to one such technique.

Sampling from the ROAR posterior occurs in two stages. In the first stage, the set of observed instance data for each action is clustered based on the similarity between states, outcomes, rewards and termination signal. That is, the observed instances induce a clustering. In the second stage, questions about particular state-actions are asked, and distributions over the clusters found in the first part, conditioned on the state-action, are generated. Once a cluster is chosen, the outcome, reward and termination signal is drawn from the outcome, reward and termination signal distributions associated with that cluster. That is, the induced clustering is used to sample hypothetical experience, generalized from the real experience.

ROAR uses a nonparametric Bayesian prior over models that can generate instances $I_t = \langle s_t, a_t, o_t, r_t, \phi_t \rangle$ representing the *state*, *action*, *outcome*, *reward* and *termination* signals for a particular step in the environment.

For a given action, the model may generate an instance by first drawing normal distributions for the state feature vector, outcome feature vector and reward signal generators, and a two-degree multinomial distribution for the

[2] In practice, the number of sweeps required is not tremendously large, but it is difficult to say anything concrete about how many are needed.

termination signal generator. These are drawn from a Dirichlet Process with a base measure that uses independent Inverse Wishart and Dirichlet distributions for drawing normal and multinomial distributions, respectively. The values making up the instance itself are then drawn from the generators.

The Inverse Wishart distribution is chosen to be the prior for the Normal distribution covariances because, as well as providing an analytical way to integrate out key parameters (Equation 2.66), the way in which an Inverse Wishart distribution is sampled is analogous to the way the maximum likelihood estimation of covariance is derived.

The basic form of the model is similiar to that described by the Infinite Gaussian mixture model [24], except in its choice of priors (Rasmussen uses the Gamma prior, rather than the Inverse Wishart), and the fact that the Infinite Gaussian mixture model draws hyperparameters from their own priors, where in this work they are fixed. Having fixed hyperparameters makes the model less flexible, but also easier to deal with both mathematically and in the approximation process.

The Chinese Restaurant Process (CRP) formulation of the Dirichlet Process is used:

$$\begin{aligned}
C = \langle C_0, C_1, ..., C_T \rangle &\sim \text{CRP}(\alpha) \\
\Sigma_i^{\text{s}} &\sim W^{-1}(\Psi_{\text{s}}, m_{\text{s}}) \\
\Sigma_i^{\text{o}} &\sim W^{-1}(\Psi_{\text{o}}, m_{\text{o}}) \\
\Sigma_i^{\text{r}} &\sim W^{-1}(\Psi_{\text{r}}, m_{\text{r}}) \\
\theta_i &\sim Dir(\langle \alpha_\phi, \beta_\phi \rangle) \\
\mu_i^{\text{s}}, \mu_i^{\text{o}}, \mu_i^{\text{r}} &\sim \text{Uniform} \\
s_t &\sim N(\mu_{C_t}^{\text{s}}, \Sigma_{C_t}^{\text{s}}) \\
o_t &\sim N(\mu_{C_t}^{\text{o}}, \Sigma_{C_t}^{\text{o}}) \\
r_t &\sim N(\mu_{C_t}^{\text{r}}, \Sigma_{C_t}^{\text{r}}) \\
\phi_t &\sim \text{Multinomial}(\theta_{C_t}).
\end{aligned}$$

A vector of cluster indices C is drawn from the CRP, and for each of these clusters the parameters are chosen from their respective priors.

Cluster Sampling

To sample from the ROAR posterior, we must sample an assignment of clusters to instances. First, the observed instances are grouped by action and the different groups are handled separately. We will refer to D as the entire collection of observed instance data for the action under consideration, and η as the collection of hyperparameters $\alpha, \Psi_{\text{s}}, m_{\text{s}}, \Psi_{\text{o}}, m_{\text{o}}, \Psi_{\text{r}}, m_{\text{r}}, \alpha_\phi, \beta_\phi$. The distribution over clusters, conditioned on observed data is

$$P(C|D, \eta) \propto P(D|C, \eta)P(C|\eta),$$

where $P(C|\eta) = P(C|\alpha)$ is the CRP prior from Equation 1.56.

We refer to $P(D|C,\eta)$ as the clustering likelihood, or the probability that the instances in D are clustered according to C. An important assumption is independence across clusters, so we know that $P(D|C,\eta) = \prod_i P(D^i|\eta)$, where $D^i = \{D_t|C_t = i\}$ is the collection of instances assigned to cluster i.

The i^{th} cluster likelihood $P(D^i|\eta)$ is then split up into its four components for the state, outcome, reward and termination signals. Since they are assumed to be independent,

$$P(D^i|\eta) \quad = \quad P(s^i|\Psi_{\mathrm{S}}, m_{\mathrm{S}})P(o^i|\Psi_{\mathrm{O}}, m_{\mathrm{O}})P(r^i|\Psi_{\mathrm{r}}, m_{\mathrm{r}})P(\phi^i|\alpha_\phi, \beta_\phi).$$

To find the likelihood that some set X of real vectors was drawn from a normal distribution whose mean is the sample mean and whose covariance Σ is drawn from an Inverse Wishart prior, we can use the fact that the Inverse Wishart is conjugate prior to the normal distribution to integrate out the parameter Σ:

$$\begin{aligned} P(X|\Psi,m) &= \int_\Sigma N(X|\bar{X},\Sigma)W^{-1}(\Sigma|\Psi,m)d\Sigma \\ &= \frac{1}{\pi^{\frac{np}{2}}} \frac{\Gamma_p\left(\frac{m+n}{2}\right)}{\Gamma_p\left(\frac{m}{2}\right)} \frac{|\Psi|^{\frac{m}{2}}}{|\Psi+S|^{\frac{m+n}{2}}}, \end{aligned} \tag{2.66}$$

where n is the number of vectors in X, p is the dimensionality of the data, the scatter matrix $S = \sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})^T$, the mean $\bar{X} = \frac{1}{n}\sum_{i=1}^n X_i$, and $\Gamma_p(z) = \pi^{p(p-1)/4}\prod_{j=1}^p \Gamma\left(z + \frac{1-j}{2}\right)$.

Similar techniques can be used to find the likelihood that some data came from a multinomial whose parameter θ is drawn from a Dirichlet prior:

$$\begin{aligned} P(x|\alpha) &= \int_\theta Mult(x|\theta)Dir(\theta|\alpha)d\theta \\ &= \frac{(\sum_{i=1}^n x_i)!}{\prod_{i=1}^n x_i!} \frac{\Gamma(\sum_{i=1}^n \alpha_i)}{\prod_{i=1}^n \Gamma(\alpha_i)} \frac{\prod_{i=1}^n \Gamma(x_i+\alpha_i)}{\Gamma(\sum_{i=1}^n x_i+\alpha_i)}. \end{aligned}$$

Parameter Sampling

Given a particular clustering assignment C, the parameters for individual clusters are independent. For the continuous features (*state*, *outcome* and *reward*) the normal parameters can be sampled from the Inverse Wishart posterior. For the discrete feature (*terminal* indicator) the Multinomial parameters can be sampled from the Dirichlet posterior. Through this process of finding a clustering and using posterior sampling to find cluster parameters, a sample is drawn from the ROAR posterior.

Instance Sampling and Trajectory Sampling

Once a model M is sampled from the ROAR posterior, simulating experience is straightforward. Outcomes, rewards and termination indicators $\langle o, r, \phi \rangle$ can be sampled from the model, conditioned on some particular state and action $\langle s, a \rangle$. Since $\langle o, r, \phi \rangle$ is independent of $\langle s, a \rangle$, given a cluster, the algorithm must first choose such a cluster c according to $P(c|s, a, M) \propto P(s|c, M)P(c|M, a)$. The outcome, reward and terminal indicator are then sampled individually according to $P(o|c, M)$, $P(r|c, M)$ and $P(\phi|c, M)$.

It is important to note that the cluster c chosen in this step does not have to be one of those created during the inference process. The CRP always leaves non-zero weight for creating a new cluster, and new clusters (drawn according to the Inverse Wishart and Dirichlet prior) must be allowed. The ability to reason about new clusters is especially important when the agent wants to make a prediction about a state that is very far from those states already observed. In this case, the likelihood of this state coming from one of the existing clusters will shrink (as it is farther from the means of these clusters) and a new, completely hypothetical cluster will be created.

For planning, dealing directly with the Gaussian mixture model described in this section is inconvenient. It is much simpler to sample transitions directly, using Monte-carlo simulations, rather than deal with their distributions. To generate n instances from the model, we sample from the distribution

$$P(I_1, ..., I_n | M, s_1, a_1, ..., s_n, a_n)$$

. That is, we jointly draw a collection of new instances $I_t = \langle s_t, a_t, o_t, r_t, \phi_t \rangle$, conditioned on the states and actions that make them up. By doing so, we can generate the outcomes, rewards and termination signals necessary for our planner.

The distribution can be broken down according to specific instances to be

$$\prod_{j=1}^{n} P(I_j | M, I_1, ..., I_{j-1}, s_j, a_j).$$

That is, the distribution of an instance I_j depends on all previous instances drawn: each future sample is conditioned on the previous samples, as if they were observations. This sampling process creates some consistency in parts of the state space with little or no data.

Although it is tempting to sample the instances i.i.d., this shortcut will lead to all instances in unexplored areas of the state space being sampled directly from the prior[3]. Doing so causes the unexplored areas to appear the same as the other unexplored areas in every model sampled. When they are sampled jointly as described, the *distributions* for those unexplored areas are sampled from the prior, but the *instances* are sampled from those distributions. As a result the unexplored areas are distinct from one another and from those in other

[3] In the generative model, the means for new cluster outcome and reward distributions are drawn from a uniform distribution. In practice, using a mean of zero for the first sample, and the sample mean thereafter, is effective.

models. There is same-model consistency in these areas that is not achieved otherwise. Figure 2.1 shows how underrepresented parts of the state-space can have consistent estimates.

2.4 Sample Complexity

Since a common subgoal for a reinforcement-learning agent is to learn a model, it is useful to quantify the sample complexity of a model in the first place. It is certainly true that there are some combinations of model priors and "true" models[4] that do not allow learning of the model without an unreasonably large number of samples.

Example 1. Consider the coin flipping experiment in Section 1.2.1. Using a variable x instead of the concrete likelihood $\frac{9990}{10000}$, we get the model

$$\phi = \begin{cases} 0.5 & \text{w.p. } x, \\ 1 & \text{w.p. } 1-x, \end{cases} \tag{2.67}$$

$$\rho \sim \phi, \tag{2.68}$$

$$H \sim \text{Binomial}(\rho, n). \tag{2.69}$$

For any given probability in Equation 1.42, there is a number of heads-in-a-row, without any tails, that is required before the posterior indicates that the coin is more likely biased than not. The threshold will occur when $P(\rho = 1|H = n, n) = P(\rho = 0.5|H = n, n)$. If we substitute x for $\frac{9990}{10000}$ in Equation 1.42, this threshold gives us the relation

$$x = \frac{1}{2^n + 1}. \tag{2.70}$$

[4] That is, models from which the observations are sampled.

Given this relation, we can say something the sample complexity of a biased coin is with this prior.

With the coin flipping example, we give only two possibilities in the prior: double-headed or unbiased. These two coins are not ϵ-close to each other, for any reasonable ϵ, since the likelihood of *heads* for the double-headed coin is 0.5 away from that of the unbiased coin, and the same distance for tails. If the posterior has a probability of at least δ of sampling an unbiased coin after B flips, the double-headed coin has a sample complexity greater than B, with respect to the prior in Equation 2.67.

Choosing $\epsilon < 0.5$, so that an unbiased coin is not considered an ϵ-approximation of a double-headed coin, there is a relation between the number of samples B, the prior likelihood x, and the probability of failure δ. We need to find B such that the posterior likelihood for the double-headed coin is at least $1 - \delta$. That is,

$$\begin{aligned}
\frac{P(\rho = 1 | H = B, n = B)}{P(\rho = \frac{1}{2} | H = B, n = B)} &\geq \frac{1-\delta}{\delta}, && (2.71)\\
\frac{\binom{B}{B} 1^B 0^0 (1-x)}{\binom{B}{B} \frac{1}{2}^B \frac{1}{2}^0 x} &\geq \frac{1-\delta}{\delta}, && (2.72)\\
\frac{(1-x)}{\frac{1}{2^B} x} &\geq \frac{1-\delta}{\delta}, && (2.73)\\
2^B (1-x)\delta &\geq (1-\delta)x, && (2.74)\\
2^B &\geq \frac{(1-\delta)x}{(1-x)\delta}, && (2.75)\\
B &\geq \log_2\left(\frac{(1-\delta)x}{(1-x)\delta}\right). && (2.76)
\end{aligned}$$

Then, if we set $x = 0.999$ and $\delta = 0.001$, we find that $B \geq 20$ is sufficient.

The rest of this section will attempt to formalize the concept of sample complexity for Bayesian machine learning.

2.4.1 Sample Complexity of the Model for a Stochastic Function

Given

- a set X,
- a set Y,
- a similarity metric $\sigma : X \times X \to [0,1]$,
- any base value $x_0 \in X$,
- any vector of example values $\bar{x} \in X^n$ such that $\sum_{i=1}^{n} \sigma(x_0, \bar{x}_i) \geq B$,
- a set of models P with $\eta_0 \in P$,
- a stochastic function $f_{\eta_0} : X \to \Pi(Y)$,
- and some prior distribution $\phi \in \Pi(P)$ over models,

we say that a specific model $\eta_0 \in P$ has a sample complexity of B with ϕ if the process

$$\bar{y}_i \sim f_{\eta_0}(\bar{x}_i), \tag{2.77}$$

$$\tilde{\eta} \sim \phi|(\bar{x}, \bar{y}), \tag{2.78}$$

results in the condition

$$\forall_{y \in Y} |f_{\tilde{\eta}}(y|x_0) - f_{\eta_0}(y|x_0)| \leq \epsilon \tag{2.79}$$

holding with probability at least $1 - \delta$.

Note that this must hold true for any $\bar{x}$, and that this $\bar{x}$ is chosen beforehand.

This guarantee can easily be adapted for a discrete set X whose members cannot be generalized. In this case, the similarity metric σ is the Kronecker delta, where the result is 1 if the two values are identical, and 0 otherwise. With this metric, there must be at least B examples of x_0 in the set of example values.

To map this setting to the coin-flip example above, consider the set X to be the set of all coins, double-headed or not. Then, we need examples of flips from the coins in that set such that the coins are sufficiently similar to the one we're thinking about, x_0. In this case, we only care about coins that are exactly the coin we're trying to estimate, so we need B examples of that coin. The set Y will be the set $\{\text{heads}, \text{tails}\}$, and η_0 is a model indicating that coin x_0 is double-headed. As a result, flipping the double-headed coin, or calling $f_{\eta_0}(x_0)$, should result in heads each time.

Intuitively, if a model has a sample complextiy of B, then we only need B evidence (according to the similarity metric σ) at a given point to have a posterior sample be ϵ-accurate at that point with probability $1 - \delta$.

In Chapter 4, I will show that this sample complexity guarantee is a sufficient condition for the posterior accuracy conditions in the analysis of a model-based Bayesian reinforcement-learning algorithm. In the later sections of this chapter, I will describe how to fit different types of MDPs, and their priors, to this guarantee.

2.4.2 Sample Complexity of the Transition Function for a Discrete-state, Discrete-action MDP

An MDP m_0 can be considered the model for a stochastic function. The transition function $T_{m_0} : S \times A \rightarrow S$ is a function using the MDP components as its model. Specifically, $T_{m_0}(s,a)$ is a multinomial distribution parameterized by an unknown $\theta_0^{s,a}$ vector, which has a prior distribution ϕ. The goal is to then evaluate the sample complexity of a particular vector $\theta_0^{s,a}$ is, with samples from the stochastic function $T_{m_0}(s,a) = \text{Mult}(\theta_0^{s,a})$.

Given

- X is the set of all state-action pairs $S \times A$,
- Y is the set of states S,
- x_0 is some state-action pair (s_0, a_0),
- $\bar{x}$ is any vector of state-action pairs that contains at least B instances of (s_0, a_0),
- P is the set of all possible multinomials for each of the state-action pairs in the MDP,
- and f_{η_0} is T_{m_0}, or the transition function of the true MDP m_0,

when the process

$$s'_i \sim \text{Mult}(\theta_0^{s_i a_i}), \tag{2.80}$$

$$\tilde{\theta}_{s,a} \sim \phi | (s_1, a_1, s'_1), (s_2, a_2, s'_2), ..., (s_n, a_n, s'_n), \tag{2.81}$$

results in a posterior sample satisfying

$$\forall_{s' \in S} |\tilde{\theta}_{s_0 a_0}(s') - \theta^0_{s_0 a_0}(s')| \leq \epsilon, \tag{2.82}$$

with probability at least $1 - \delta$, we can say that the MDP m_0's transition function has a sample complexity of B.

Sample Complexity of an MDP Using the Flat Dirichlet Multinomial

The Flat Dirichlet Multinomial prior, or FDM, described in Chapter 2, is one of the simplest priors for a discrete-state, discrete-action MDP. With the FDM prior, the transition function for each state is treated as an independent problem where the parameter θ_0 is learned from a set of samples from $\text{Mult}(\theta_0)$, and $\theta \sim \text{Dir}(\alpha)$.

We can then simplify the analysis of a model with the FDM prior's sample complexity to that of the sample complexity of some random simplex θ_0 with a Dirichlet prior.

Example 2. Let $\alpha = (1,1)$ and $\theta_0 = (\frac{1}{2}, \frac{1}{2})$. This problem is the same as the sample complexity of an unbiased coin with a uniform prior over its bias (that is, an unbiased coin is just as likely as a double-headed coin, which is just as likely as a coin that lands heads 2/3 of the time, etc.).

We can resolve the question of the sample complexity for this prior by answering the question of how many times we need to flip this biased coin such that our posterior samples are accurate.

Let $\epsilon = 0.1$ and $\delta = 0.1$. Then, for the unbiased coin to have a sample complexity of B, we need to choose a B that is high enough so that 90% of the time when we perform the experiment of flipping the coin B times and sampling from the posterior, we get a coin with a heads likelihood in $[0.4, 0.6]$ and a tails likelihood in $[0.4, 0.6]$.

Since the Multinomial distribution with two outcomes is the Binomial distribution, and the Dirichlet distribution on the two-dimensional simplex is the

Beta distribution, we can use those distributions in our analysis. That is, given the process

$$\begin{aligned} H &\sim \text{Bin}(\rho = 0.5, B), && (2.83) \\ \tilde{\rho} &\sim \text{Dir}(\alpha = (H+1, \beta = B - H + 1)), && (2.84) \\ & && (2.85) \end{aligned}$$

choose a value B such that

$$P(0.4 \leq \tilde{\rho} \leq 0.6) \geq 1 - \delta, \quad (2.86)$$

$$\begin{array}{l} \sum_{H=0}^{B} \text{Bin}(H|\rho = 0.5, n = B) \\ \cdot \int_{\tilde{\rho}=0.4}^{0.6} \text{Beta}(\rho|\alpha = H+1, \beta = B - H + 1), \end{array} \geq 1 - \delta \quad (2.87)$$

$$\sum_{H=0}^{B} \binom{B}{H} \frac{1}{2^B} \left[\begin{array}{l} I_{\min(1,\frac{1}{2}+\epsilon)}(H+1, B-H+1) \\ -I_{\max(0,\frac{1}{2}-\epsilon)}(H+1, B-H+1) \end{array} \right] \geq 1 - \delta, \quad (2.88)$$

where $I_p(\alpha, \beta)$ is the regularized incomplete gamma function, and $\int_0^p \text{Beta}(p|\alpha, \beta)dp = I_p(\alpha, \beta)$.

The smallest B that satisfies Equation 2.88 can be found numerically, and happens to be 21. As a result, the unbiased coin has a sample complexity of $B = 20$ with the $\text{Beta}(1, 1)$ prior, for $\epsilon = 0.1$ and $\delta = 0.1$.

In general, for any hyperparameter α, set of states S, and true model θ_0,, we can relate B, ϵ and δ with the following equation:

$$\sum_{\bar{s'}:||\bar{s'}||_1 = B} \text{Mult}(\bar{s'}|\theta_0, B) \int_{\theta:||\theta-\theta_0||_1 \leq \epsilon} \text{Dir}(\theta|\alpha + \bar{s'})d\theta \geq 1 - \delta. \quad (2.89)$$

Although the analysis is difficult, numerical techniques can be used to find the smallest value B for a given ϵ and δ.

2.4.3 Sample Complexity of the Prior

So far, this chapter has discussed the sample complexity of a particular "true" model for a given prior.

That is, if we start with ϵ, δ, ϕ, and η_0, we can imagine a function $B(\eta_0, \phi, \epsilon, \delta)$ that tells us how many samples are required to learn that model, given our accuracy constraints, desired likelihood of success, and prior distribution.

It also makes sense to talk about the expected sample complexity, given a prior. The sample complexity of a prior is

$$L(\phi, \epsilon, \delta) = \int_{\eta_0} \phi(\eta_0) B(\eta_0, \phi, \epsilon, \delta) d\eta_0. \tag{2.90}$$

In other words, the sample complexity of a prior is the expected sample complexity of a model sampled from that prior.

Although the sample complexity is difficult to express exactly, it is related to the probability of success. If, with $\bar{x} : \sum_i \sigma(x_0, \bar{x}_i) \geq B$,

$$\int_{\eta_0} \phi(\eta_0) \left[\int_{\bar{y}} f_{\eta_0}(\bar{y}|\bar{x}) \left[\int_{\tilde{\eta}} \phi(\tilde{\eta}|(\bar{x}, \bar{y})) \mathbb{1}(||\tilde{\eta} - \eta_0|| \leq \epsilon) \, d\tilde{\eta} \right] d\bar{y} \right] d\eta_0 \geq 1 - \delta. \tag{2.91}$$

In english, first we pick a "true" model η_0 according to the prior ϕ. Then, we make some observations $\hat{y}$ from the world created by η_0. Once we have the "true" model and the observations, we see how many models are ϵ-close to the "true" model, weighted according to their posterior likelihoods.

Example 3. We can find the sample complexity of the Beta-Binomial model,

$$\rho \sim \text{Beta}(\alpha, \beta), \tag{2.92}$$

$$H \sim \text{Bin}(\rho_0, B). \tag{2.93}$$

Table 2.1: Sample complexity for the Beta-Binomial. Using the model from Section 3, the posterior for the latent variable ρ is ϵ-accurate with probability at least $1-\delta$ if B is above some threshold. This table gives examples of the relations between the model hyperparameters, ϵ, δ, and B.

(α, β)	δ	ϵ	lowest B
$(1,1)$	0.1	0.1	91
$(50,50)$	0.1	0.1	31
$(100,100)$	0.1	0.1	0
$(1,5)$	0.1	0.1	61
$(5,25)$	0.1	0.1	43
$(10,50)$	0.1	0.1	13

The Beta-Binomial prior's ρ parameter has a sample complexity of B if

$$\begin{aligned} &\int_{\rho_0} \mathrm{Beta}(\rho_0|\alpha,\beta) \textstyle\sum_{H=0}^{B} \mathrm{Bin}(H|\rho_0, B) \\ &\cdot \int_{\tilde{\rho}=\rho_0-\epsilon}^{\rho_0+\epsilon} \mathrm{Beta}(\tilde{\rho}|\alpha+H, \beta+B-H) d\tilde{\rho}\, d\rho_0 \end{aligned} \quad \geq \quad 1-\delta \tag{2.94}$$

$$\begin{aligned} &\int_{\rho_0} \mathrm{Beta}(\rho_0|\alpha,\beta) \textstyle\sum_{H=0}^{B} \mathrm{Bin}(H|\rho_0, B) \\ &\cdot \left[I_{\rho_0+\epsilon}(\alpha+H, \beta+B-H) - I_{\rho_0-\epsilon}(\alpha+H, \beta+B-H) \right] d\rho_0 \end{aligned} \quad \geq \quad 1-\delta \tag{2.95}$$

where $I_x(a,b) = I_{\max(0,\min(1,x))}(a,b)$ is the regularized incomplete gamma function.

For a given ϵ, δ, α, and β, the smallest B that satisfies Equation 2.95 can be found numerically. Table 2.1 lists several example sample complexities for different values of ϵ and δ, and different prior parameters.

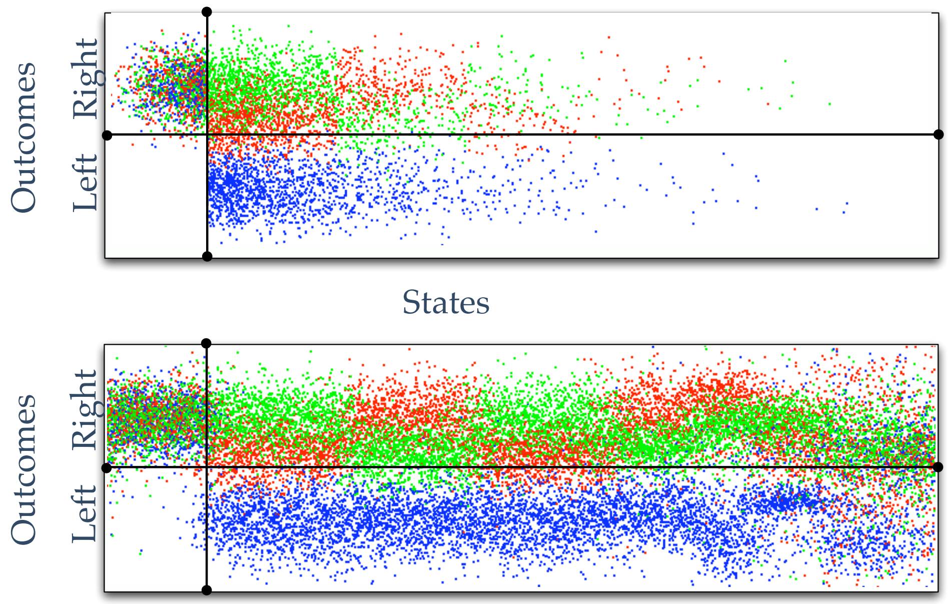

Figure 2.1: *Riverswim* is a simple environment where a fish tries to make its way up the river, and can take one of three actions: red, green, or blue. In these figures, the horizontal axis represents a state in the environment: further to the right means further up the river. The vertical axis represents a possible outcome: further up means that an action took the fish futher up the river, with a point on the origin meaning that the fish did not move. Each dot represents an action taken, with its horizontal position being where the fish was, and the vertical position being how far it moved. The top figure is experience in the actual environment; every dot represents an action that the agent took, and feedback it got from the environment. The bottom figure is experience sampled from the ROAR posterior. As a human, one can see that it matches very closely the dynamics from the original environment, especially where there are more samples. Where there are fewer samples, to the right, *ROAR* has less to go on and its estimates are less accurate. However, because each sample is conditioned on previous samples, there are still visible clusters to the right. *ROAR* is making these up, and although they may be inaccurate, they are self-consistent and can be used to speculate about unexplored parts of the statespace without simply indicating that they are all drawn identically from the prior.

Chapter 3
Related Work

In this chapter, we discuss important related work that will influence, directly and indirectly, later contributions. Along with algorithms of general interest to a student of model-based Bayesian reinforcement-learning, there are

- existing model-based RL algorithms used for comparisons in Chapter 6 [4, 25].
- algorithms that make use of a flexible prior [9, 18, 20, 21],
- model-based Bayesian RL algorithms with PAC-BAMDP guarantees [8, 9, 21].
- the **FSSS** algorithm, which is used as a building block in Chapter 3, and several other tree search planners [26, 27].

3.1 RMAX

The **RMAX** [25] algorithm is a simple and robust method to balance exploration and exploitation in the reinforcement-learning setting. The original algorithm addresses only discrete state- and action-spaces, but the ideas apply more generally [4, 28].

To effectively use **RMAX**, the agent designer needs a metric to determine whether or not a particular state-action pair is *known*. In the discrete state and action case, the agent can count how many times it has tried a given action pair. If that number exceeds a predetermined threshold, the given pair is *known*. Otherwise, it is *unknown*.

To guide the **RMAX** agent, a model is constructed from the agent's observations, with unknown state-action pairs transitioning only to a nirvana state, or a state with the highest possible value. Known states have the transition dynamics given by the MLE. The agent then acts greedily according to this model.

Clamping unknown states at very high value will draw the agent towards these states if there is any chance that visiting those states is the optimal thing to do.

Because the model maintained by **RMAX** is accurate (when the state is known), optimistic (when the state is unknown) and there is a bounded number of times that the agent can be surprised (because there is a finite number of state-action pairs that can become known), the authors were able to show that the algorithm is PAC-MDP. That is, it will make only a small number of suboptimal decisions over the course of its lifetime.

3.1.1 Potential-based RMAX

As it stands, the **RMAX** algorithm cannot make use of any prior knowledge. All unknown states are assigned the highest possible value, and the agent is drawn to each of them in a way that depends only on how quickly it can reach them.

The **Potential-based shaping RMAX** [29] algorithm addresses this deficiency by incorporating a potential function $\phi(s)$. The agent will perform well if this potential function is *admissible*. That is, $\forall_s \; V(s) \leq \phi(s)$; the potential function's value for a given state must never be less than the true value for the state. This admissibility requirement is common in heuristic-based search algorithms, such as A* [30].

This algorithm works by assigning unknown states a value of $\phi(s)$, rather than $V_{\max}$, the highest possible value. Setting $\phi(s) = V_{\max}$ is allowed, since $V_{\max}$ automatically satisfies the admissibility requirement, but lower values result in faster convergence.

On the downside, this algorithm is very conservative, like the rest of the **RMAX** family. Requiring a fixed number of visits to a state-action pair before it becomes known means a lot of exploration steps if the heuristic is too optimistic, some of which might not be necessary in practice.

3.1.2 Fitted-RMAX

Instance-based methods represent functions by a set of examples and use a predefined similarity function to generalize to novel inputs. In the reinforcement-learning setting, an instance is the tuple $\langle s, a, s', r \rangle$, or the combination of state, action, next state and reward.

Over the course of its lifetime, the agent gathers a large set of $\langle s, a \rangle \rightarrow \langle s', r \rangle$ mappings. When it observes a new state-action pair, it can compare it to those it has seen in the past and predict a next-state and reward that fits with those already observed.

The ideas behind instance-based model learning can be combined with the **RMAX** algorithm to create **Fitted-RMAX** [4]. **Fitted-RMAX** uses a Gaussian kernel to determine the similarity between two states:

$$d(s_1, s_2) = \exp\left(\frac{-||s_1 - s_2||_2^2}{b}\right), \qquad (3.1)$$

where b is a *width* parameter used to determine how much distance affects the similarity between two states.

The possible outcomes and rewards are then taken from a weighted average of those seen before when taking the action in question, where each possibility $\langle o_i, r_i \rangle$ is weighted by $d(s_t, s_i)$, where s_t is the state from which the agent is considering taking an action.

Since this algorithm is applied in continuous domains, the standard definition of knownness that uses visit counts cannot be used. Instead, a distance metric needs to be incorporated. The sum of the similarity scores between the state in question and all previously experienced states is taken and compared to a threshold B, over which the knownness metric declares a state to be known. Unknown states are then assigned maximum value, according to the rules of **RMAX**.

Once a policy is found for the **Fitted-RMAX** model, using some existing continuous-state planner, the agent will be drawn to explore states that are either unknown or have high value, like the original **RMAX** on which it is based.

3.2 Bayes-adaptive Markov Decision Processes

The Bayes-adaptive MDP, or BAMDP [7], is an MDP whose optimal policy is the Bayes-optimal policy for some other unknown MDP that is drawn from a known prior. Every state in the BAMDP is a belief-state composed of a history, or the set of observations made by the agent so far, and the concrete state that the agent is currently in.

The transition function of the BAMDP is defined as follows:

$$P(\langle s', h + (s,a,s',r)\rangle, r | \langle s,h\rangle, a) = \int_M P(s', r|s,a,M)\phi(M|h)dM, \qquad (3.2)$$

where s and s' are *concrete* states from the unknown MDP, $\langle s, h\rangle$ is a belief-state, in the BAMDP, where the agent begins, and is composed of the concrete state s and history h. The value h is the summary of all transitions observed by the agent over the course of its lifetime. Here, they are represented by (s,a,s',r) tuples detailing the beginning and ending concrete states, the action taken and the reward received.

The likelihood of the transition in the BAMDP is taken by integrating out the underlying and unknown "true" MDP, according to MDP's posterior distribution. If a next-state sampler is needed, rather than an exact transition probability, the process is to first sample an MDP M from the posterior distribution $\phi(M|h)$, and then sample the next state s' and reward r from M.

By constructing the BAMDP according to Equation 3.2, the optimal policy is identical to the Bayes-optimal policy in the learning setting. This fact follows because the succession of transition likelihoods is exactly the same as those indicated in the Bayesian model[1].

Unfortunately, the BAMDP has potentially an infinite number of states, since each possible history maps to a different belief state. However, as the number of observations from a particular concrete state and using a particular action grows large, the distribution over the possible next concrete states converges, allowing a clever algorithm designer to collect large sets of belief states together into equivalent classes, for the purposes of planning. An easy way to make this optimization is to not adjust the posterior when an observation

[1] See Section 1.1.4 for a full explanation of why optimal in the BAMDP is Bayes-optimal while learning the MDP.

```
**Input**: initial state $s$, prior $\Phi$, #steps $K$
$O \leftarrow \{\}$
**for** *ever* **do**
    $M \sim \Phi|O$
    $\pi \leftarrow \text{solve}(M)$
    **for** $K$ *times* **do**
        $a \leftarrow \pi(s)$
        $s', r \leftarrow \text{perform}(a)$
        $O \leftarrow O \cup \{(s, a, s', r)\}$
        $s \leftarrow s'$
```

Algorithm 2: Bayesian DP(s, Φ, K)

is made for a particular state-action pair s, a if that pair has been observed at least B times in the past, for a choice of B sensible for the prior. This trick can be used in any model-based Bayesian RL algorithm that uses a posterior sampler.

3.3 Bayesian Dynamic Programming

Bayesian Dynamic Programming [20], or **Bayesian DP**, is one of the first model-based reinforcement learning algorithms to take advantage of posterior sampling. The agent requires a model prior ϕ and a parameter K, which tells the agent how many steps should be taken between samples.

Algorithm 2 lays out the **Bayesian DP** algorithm. The agent starts in some initial state with no information about the environment except that which can be gleaned from the prior Φ. Before taking any actions, the agent samples a model M from the posterior $\Phi|O$, where Φ is the model prior and O is the set of observations recorded so far. Once M is sampled from the posterior, it is solved optimally by whatever planning algorithm the designer prefers. The resulting policy π is acted upon for the next K steps, and each step is recorded and added to the set of observations. After K actions have been performed, a new model is sampled from the updated posterior, and the process is repeated until the end of the experiment.

The act of periodically sampling from the posterior is intended to give the agent glimpses of possible worlds. Some of the times a sample is drawn, a state that the agent has not visited much will appear better than it actually is, drawing the agent towards this relatively unknown state. On the other hand, some times that unknown state will appear to have a very low value, causing the agent to avoid it. State-action pairs that the agent has experienced many times in the past will have relatively stable dynamics, and will usually have a consistent value with respect to its neighboring states. As a result, this approach strikes a balance between exploration of states in which it is uncertain and exploitation of states in which it is confident.

The fact that some samples will make an unknown state appear attractive and other samples will make the same unknown state appear unattractive requires **Bayesian DP** to be careful about when and how often it samples new models. Too long between samples, and the agent is acting based on stale information. Too short between samples and the agent can end up dithering, as explained next.

The parameter K, which is the number of steps taken between samples, should be chosen to approximate the length of a trial (in situations where there is a goal state) or the width of the MDP (maximum expected number of steps to get from any given state to any other given state) when there is an infinite horizon.

The idea here is that the agent should be able to reach any particular state it wants before the next sample is taken and the agent might change its mind about where it is trying to go. Consider a simple *chain* MDP, where only the two states on the end are unknown. In a given posterior sample, one of the two ends may have a higher apparent value than the other. In the next sample, the relative values of these two states may switch. If a new sample is taken before the agent can reach one of the ends (and therefore learn about its dynamics), thrashing can occur, resulting in an exponential number of steps (relative to the number of states) before knowledge is gained.

To understand how thrashing can cause an exponential number of sub-optimal steps, consider an MDP with one root state and N "chains" leading off from that root state. At the end of one of these chains is the goal, and the prior does not give preference to which chain might have that goal. At the root, the agent can choose which chain to travel, but after that is limited to two choices: continue or reset. Everything about the MDP is known except the identity of the chain containing the high-valued goal. As a result, the posterior does not change except when the agent reaches the end of a chain that it has not reached before. Since the prior has no indication of which chain has the goal, each time a sample is drawn the goal is randomly put at the end of one of the chains. With **Bayesian DP**, if the agent is already on this chain, it can continue getting closer to the goal. If the agent is on the wrong chain, it needs to reset and head down the correct one.

If the length of each of the chains is L, this requires L/K identical guesses in a row in order for the agent to either get the goal or remove that chain from the posterior. With N unknown chains remaining, the likelihood of this event happening for any $\frac{L}{K}$ consecutive posterior samples is $(\frac{1}{N})^{\frac{L}{K}}$. Briefly, consider the problem of counting the number of flips of a biased coin before M consecutive heads turn up, where each flip is heads with probability p. The likelihood of flipping M heads in a row is p^M. At the beginning, or immediately following a tails, there are M different events that can happen, each with different likelihoods. There are $M-1$ versions of flipping T heads and then a tails, for $0 \leq T < M$ (a failure), each with probability $p^T(1-p)$ and taking T flips, and there is flipping M heads (a success), with probability p^M and taking M flips. Since the expected number of failures before a success is $1/(p^M)$, we can count the total number of expected steps by multiplying $1/(p^M)$ by the expected number of steps in a failure, which is

$$\begin{aligned}
E[T] &= \sum_{T=0}^{M-1} P(T)T, && (3.3)\\
&= \sum_{T=0}^{M-1} p^T(1-p)T, && (3.4)\\
&= (1-p)p \sum_{T=0}^{M-1} p^{T-1}T, && (3.5)\\
&= (1-p)p \sum_{T=0}^{M-1} \frac{dp^T}{dp}, && (3.6)\\
&= (1-p)p \frac{d\sum_{T=0}^{M-1} p^T}{dp}, && (3.7)\\
&= (1-p)p \frac{d\frac{1-p^M}{1-p}}{dp}, && (3.8)\\
&= (1-p)p \left[(1-p)(1-p^M) - \frac{Mp^{M-1}}{1-p}\right], && (3.9)\\
&= p(1-p)^2(1-p^M) - Mp^M. && (3.10)
\end{aligned}$$

Returning to the N-chain MDP, if we consider $p = (\frac{1}{N})^{\frac{L}{K}}$ and $M = \frac{L}{K}$, the expected number of samples drawn (coins flipped) in a failure will be $E[T]+1$ (the first sample is what we choose to think of as heads - the others then need to match it). With an expected $(\frac{1}{N})^{-\frac{L}{K}}$ failures before the first success, that gives an expected

$$E[\text{number of failures}]E[\text{steps per failure}] + E[\text{steps per success}] \quad (3.11)$$

$$= \left(\frac{1}{N}\right)^{-\frac{L}{K}} (p(1-p)^2(1-p^{\frac{L}{K}}) - \frac{L}{K}p^{\frac{L}{K}} + 1) + \frac{L}{K} \quad (3.12)$$

samples before reaching the end of one of the chains. Ignoring the smaller pieces, the $(\frac{1}{N})^{-\frac{L}{K}}$ factor is exponential with respect to the length of the chain.

Spacing out the MDP samples temporally allows the agent to visit at least one unknown (and optimistically sampled) state every K steps (assuming that any of the unknown states are optimistic in the posterior sample for those K steps). Choosing a K that is at least as great as the expected number of steps to get from any state in the MDP to any other state in the MDP alleviates the thrashing issue seen in the N-chains example above. With $K = L$, $(\frac{1}{N})^{-\frac{L}{K}} \to N$, and there is no exponential factor in the expectation.

3.4 Multi-Task Reinforcement Learning

Multi-Task Reinforcement Learning, or MTRL, considers the scenario where an agent is given a sequence of environments, each of which is drawn i.i.d. from some prior distribution. The **Hierarchical Bayesian MTRL** algorithm [22], or **HBMTRL** assumes the following generative process for creating sequences of MDPs:

$$\begin{aligned} C &\sim \text{CRP}(\alpha), && (3.13)\\ \theta_c &\sim \Theta, && (3.14)\\ m_i &\sim \phi(\theta_{C_i}), && (3.15) \end{aligned}$$

where α is the CRP concentration parameter, C is the assignment of MDPs to “classes” with C_i being the class of MDP m_i, Θ is a distribution over MDP hyper-parameters, θ_c is the hyper-parameter used for MDPs in class c, ϕ is some distribution over MDPs parameterized by an instance of θ, and m_i is an MDP in the sequence.

With this approach, the hyper-parameter distribution Θ and the MDP distribution family ϕ can be anything - the algorithm itself has no constraints on what may be used.

The process **HBMTRL** uses is similar to that of **Bayesian DP** [20], except that instead of sampling a single model from the posterior and acting according to its optimal policy, **HBMTRL** will sample several MDPs and discard all of them but the most likely. This has the effect of sampling from a peaked version of the posterior, where the likelihood ratios between two samples grows more severe. As the size of the sample set grows, the resulting version of the posterior becomes more and more peaked. In the limit as the size of sample set goes to infinity, this process will choose the mode of the posterior.

This algorithm is an example of using model-based Bayesian RL for transfer learning, where experience gathered in one MDP can inform an agent's behavior in another. The model in Equation 3.13 indicates that all MDPs can be divided into groups which share some basic qualities, represented by θ_i.

In the parlance of this dissertation, **HBMTRL** is both an algorithm and a family of priors. The algorithm is the process of sampling from the peaked posterior and acting according to the optimal policy of the result. The family of priors is those priors that use a Dirichlet process mixture model to sample MDPs.

3.5 BEETLE

Bayesian Exploration Exploitation Tradeoff in LEarning, or **BEETLE** [17] treats the task of Bayes-optimal behavior as optimal behavior in a POMDP whose hidden state is the MDP describing the transition probabilities between the observed states.

This algorithm assumes the Flat Dirichlet Multinomial prior, where each of the MDP's next-state distributions are drawn from a Dirichlet distribution.

The POMDP's state-space is $S \times \Theta$, where Θ is the set of next-state likelihood vectors, describing the entire dynamics of the MPD. The POMDP's action space is the same as that of the underlying MDP's. The transition function is $P(\langle s', \theta' \rangle | \langle s, \theta \rangle, a) = \theta^{s'}_{s,a} \delta(\theta, \theta')$, where δ is the Kronecker delta. The set of observations is the set of states from the underlying MDP. And, the reward function is known.

Since the Dirichlet distribution is conjugate prior to the multionimial distribution, belief monitoring is straightforward.

BEETLE is a point-based value iteration method for this sort of POMDP, and for every iteration it will prune the number of belief-states that it is considering in order to mitigate the natural intractibility of POMDP planning.

3.6 Bayesian Exploration Bonus

The **BEB** [8] algorithm is a computationally efficient method to achieve near Bayes-optimal behavior in discrete state- and action-spaces when using a Dirichlet prior.

The algorithm uses a flat Dirichlet multinomial (FDM) prior, which states that

$$\theta^{s,a} \sim Dirichlet(\alpha), \tag{3.16}$$

$$P(s'|s,a) = \theta^{s,a}_{s'}, \tag{3.17}$$

where α is a tuning parameter that correlates with variance in the next-state distribution.

BEB assumes the reward function is known.

To drive exploration, **BEB** adds an internal bonus $B(s,a)$ to the reward received when taking action a from state s. It acts greedily according to $\hat{Q}(s,a)$:

$$\begin{aligned} \hat{Q}(s,a) &= R(s,a) + B(s,a) + \gamma \sum_{s'} P(s'|s,a)\hat{V}(s'), && (3.18) \\ \hat{V}(s) &= \max_a \hat{Q}(s,a), && (3.19) \\ B(s,a) &= \frac{\beta}{1+n(s,a)}, && (3.20) \end{aligned}$$

where $n(s,a)$ is the number of times that action a has been taken from state s.

This algorithm is similar to Model-Based Interval Estimation (**MBIE**) [31], except that **MBIE** uses the square root of the visit count in its bonus. This difference causes the bonus term used by **MBIE** to decay more slowly resulting in slower convergence, and that extra conservativity allows **MBIE** to be a PAC-MDP algorithm, where **BEB** is a near Bayes-optimal algorithm when the prior is the flat Dirichlet multinomial.

3.7 Variance-based BEB

Variance-based **BEB**, or **vBEB** [21], is an adaptation of **BEB** that can make use of a flexible prior, or one that is not restricted to the flat Dirichlet multinomial.

The **vBEB** algorithm provides a framework for analyzing an arbitrary prior, and creating a bonus term based on the posterior variance for the state-action pair to which the bonus will be applied. In this framework, the flat Dirichlet multinomial gets the bonus term from Equation 3.20, and it will be different for other priors. With the proper bonus selected, **vBEB** is a PAC-BAMDP algorithm.

3.8 BOLT

Bayesian Optimistic Local Transition, or **BOLT** [9], makes use of a flexible prior and has PAC-BAMDP guarantees. **BOLT** works by constructing a hyper-model m' from the posterior $\phi|h$, where ϕ is the model prior and h is the agent's current history at the time that the hyper-model is constructed. This hyper-model has the same state-space, but an expanded action-space $A' = S \times A$. The transition function of the hyper-model is

$$T_{m'}(s'|s, \langle \hat{s}, a \rangle) \quad = \quad \int_m \phi(m|h + (s, a, \hat{s})^{(\eta)}) T_m(s'|s, a) dm, \qquad (3.21)$$

where $h + (s, a, \hat{s})^{(\eta)}$ is the current history augmented with η extra hypothetical transitions from s to $\hat{s}$ when choosing action a. As a result, $\phi(m|h + (s, a, \hat{s})^{(\eta)})$ is the hypothetical posterior likelihood of m after the extra transitions.

This hyper-model provides a way to introduce optimism into the partially known or unknown MDP. At least one of the states must have high value (otherwise, why bother?), and for a transition function that still has variance in the posterior, creating an action with a higher likelihood of ending up in that state allows the planner to pick and choose which states visits. Unlike **RMAX**- and **BEB**-based algorithms, which augment rewards for state-action pairs with poorly known transition functions, **BOLT** changes the transition function itself to reflect that uncertainty.

3.9 Sparse Sampling

Sparse Sampling [26], outlined in Algorithm 3, is a Monte-carlo tree construction algorithm for value estimation on MDPs. This algorithm works by performing exhaustive search, using Monte-carlo sampling to estimate transition probabilities.

```
Input: initial state $s$, next-state sampler $P$, reward function $R$, discount
        factor $\gamma$, accuracy $\epsilon$, probability of failure $\delta$
Output: estimated value $V$
$R_{\max} \leftarrow \max_{sa} R_{sa}$
$V_{\max} \leftarrow \frac{R_{\max}}{1-\gamma}$
$\lambda \leftarrow \frac{\epsilon(1-\gamma)^2}{4}$
$k \leftarrow$ #actions
$H \leftarrow \left\lceil \log_\gamma(\lambda / V_{\max}) \right\rceil$
$C \leftarrow \frac{V_{\max}^2}{\lambda^2}\left(2H \log \frac{kHV_{\max}^2}{\lambda^2} + \log \frac{R_{\max}}{\lambda}\right)$
$V \leftarrow$ Sparse Sampling Recursion$(s, P, R, \gamma, H, C)$
return $V$
```

Algorithm 3: Sparse Sampling$(s, P, R, \gamma, \epsilon, \delta)$

```
Input: initial state $s$, next-state sampler $P$, reward function $R$, discount
        factor $\gamma$, depth $d$, branching factor $C$
Output: estimated value $V$
if $d = 0$ then
    $V \leftarrow 0$
else
    for $a \in A$ do
        $Q_a \leftarrow R_{sa}$
        for $C$ times do
            $s' \sim P_{sa}$
            $V' \leftarrow$ Sparse Sampling Recursion$(s', P, R, \gamma, d-1, C)$
            $Q_a \leftarrow Q_a + \frac{\gamma V'}{C}$
    $V \leftarrow \max_a Q_a$
return $V$
```

Algorithm 4: Sparse Sampling Recursion(s, P, R, γ, d, C)

Many value-estimation algorithms, especially those preceding the publication of **Sparse Sampling**, have a strong linear or super-linear dependence on the number of states in the MDP. This dependence often arises as a result of the algorithm's attempt to estimate the value of all states in the MDP.

Sparse Sampling reduces the dependence on the number of states by only considering states that are likely to be reached from the state whose value is being estimated. This state is the root of a search tree. Each node in the tree is one particular state in the MDP (though more than one node can share a single state).

This algorithm makes use of a next-state generator P to generate transition samples. If the runtime required to sample from P_{sa} does not depend on the number of states, then **Sparse Sampling**'s runtime will also not depend on the number of states.

While this sampling runtime condition will not hold true for general MDPs, it makes practical sense for many situations. For example, in a continuous-state MDP (infinite number of states) that uses an offset drawn from a normal distribution to determine the next state, the amount of time required to sample from that normal distribution will depend only on the number of state features, rather than the number of states. Also, in a gridworld, the next state is often chosen from those states adjacent to the current state, and can be sampled very quickly.

The **Sparse Sampling** algorithm works by sampling a fixed number C of next-states for each action, when taken from the root. The value of those states is estimated recursively and with a search-depth of one less than the current search depth. As a base case, if the current search depth is zero, the value is approximated by 0.

The algorithm must be provided a probability of success $1 - \delta$, required accuracy ϵ and discount factor γ. These parameters inform **Sparse Sampling**'s choice of the parameter C and the desired search depth.

The leaf nodes in the search tree, those whose value is approximated by 0, are too far in the future, according to the discount factor γ and accuracy constraint ϵ, to have a significant affect on the value of the root.

Usually, the value C needed in order to achieve a particular ϵ and δ is too high for any kind of practical use. Simply running **Sparse Sampling** on the

BAMDP, and taking the action that it suggests has the highest value, results in an approximately Bayes-optimal agent. But, since **Sparse Sampling** would require an enormous amount of computation, algorithms that require less computation (usually at the cost of some number of sub-Bayes-optimal steps) are desireable.

3.10 Bayesian Sparse Sampling

Bayesian Sparse Sampling [18] is an adaptation of Sparse Sampling (see Section 3.9) that uses a model prior and a targetted method of choosing actions while planning.

Where Sparse Sampling is only able to estimate a state's value for a particular concrete model, Bayesian Sparse Sampling works with uncertainty, codified by a provided model prior. Every time a next-state is needed to further build the search tree, a model is sampled from the posterior, and the next-state is sampled from that model. Future next-states in the sub-tree of the newly sampled next-state will include this generated transition in their observation sets, allowing the sub-trees to simulate learning. This process is analogous to **Sparse Sampling** in the BAMDP, except that **Bayesian Sparse Sampling** explores only a small portion of the search tree that **Sparse Sampling** would discover.

The method of action-selection differs from Sparse Sampling in that rather than trying each action a number of times, Bayesian Sparse Sampling will run a series of roll-outs (simulated trajectories through the BAMDP). To choose the action at each stage in the roll-out, Bayesian Sparse Sampling samples a model from the posterior and finds the optimal action in the sampled model from the current state, and chooses that state, similar to Thompson sampling [32].

Choosing actions by identifying the optimal action in posterior samples allows Bayesian Sparse Sampling to target its exploration to attempt only actions that have a chance of being optimal. Also, when the posterior has converged to something close to the truth, the sampled model will be consistent and correct, and Bayesian Sparse Sampling will only choose the best action, wasting no samples in its value estimation.

This algorithm has some efficiency issues; it requires sampling from the posterior and solving for the true optimal action in the inner loop. Both posterior sampling and model solving can often be computationally difficult. There are cases where the posterior sampling and model solving can be done efficiently, for instance when the model is a bandit problem with the Beta prior [18].

3.11 Bayesian Approaches to Acting in POMDPs

The majority of the work discussed in this dissertation focuses on using priors to help an agent explore and exploit an MDP. Even if that MDP is not known exactly, the agent gets to know exactly what states it visits, and is guaranteed that $T(s'|s,a)$ is constant, if sometimes unknown, throughout the course of the experiment.

When trying to act in a POMDP, the agent only gets to oberve a signal, which is drawn from a distribution and conditioned on the unobserved state. This disconnect makes learning much more difficult.

The use of Bayesian non-parametric techniques is a proven method for learning latent variables in a variety of situations [1, 33, 34], and it makes sense to use them for POMDP inference as well.

The Infinite POMDP [35] is an example of using non-parametric techniques for POMDP inference. It uses a CRP to assign specific observations to states, and a Stick-breaking process to create next-state distributions. The CRP and the Stick-breaking process are both different formulations of the Dirichlet process, so the result is a hierarchical Dirichlet process.

The model can be written as follows:

$$\begin{aligned} T_{s,a} &\sim \text{Stick}(\lambda), && (3.22)\\ \Omega_{s,a} &\sim H, && (3.23)\\ R_{s,a} &\sim H_R, && (3.24)\\ S &\sim \text{CRP}(\alpha), && (3.25)\\ s_{t+1} &\sim T_{S_t,a}, && (3.26)\\ o_t &\sim \Omega_{S_t,a}, && (3.27)\end{aligned}$$

where λ and α are concentration hyper-parameters to their respective distributions, H is a prior over different observation distributions, $\Omega_{s,a}$ is the observation distribution when in hidden state s and performing action a, Rs,a is the reward distribution when in hidden state s and performing action a, H_R is a prior over different reward distributions, S is the assignment of time-steps to states (so at time t, the agent was in state S_t), and o_t is the observation made after taking an action during time-step t.

This model is very general, and allows the agent designers choose a model for the observations and rewards that fits with the domain they are working with.

```
**Input**: initial state $s$, next-state sampler $P$, reward function $R$, discount
        factor $\gamma$, **UCB** parameter $C$
**Output**: estimated value $V$
$\forall_s \; n_s \leftarrow 0$
$\forall_s \; V_s \leftarrow 0$
$\forall_{s,a} \; n_{s,a} \leftarrow 0$
$\forall_{s,a} \; Q_{s,a} \leftarrow 0$
**if** $s$ *is terminal* **then**
    $V_s \leftarrow 0$
    **return** $V_s$
**if** $n_s \neq 0$ **then**
    $\forall_a \; B_{s,a} \leftarrow 2C\sqrt{\frac{\log n_s}{n_{s,a}}}$
    $a \leftarrow \text{argmax}_a \; Q_{s,a} + B_{s,a}$
**else**
    $a \leftarrow$ randomly chosen action
$s' \leftarrow P(s,a)$
$r \leftarrow R(s,a)$
$V_{s'} \leftarrow \text{UCT}(s', P, R, \gamma, C)$
$\hat{Q}_{s,a} \leftarrow r + \gamma v_{s'}$
$Q_{s,a} \leftarrow (n_{s,a}Q_{s,a} + \hat{Q}_{s,a})/(n_{s,a}+1)$
$n_s \leftarrow n_s + 1$
$n_{s,a} \leftarrow n_{s,a} + 1$
$V_s \leftarrow \max_a Q_{s,a}$
**return** $V_s$
```

Algorithm 5: UCT(s, P, R, γ, C)

3.12 Upper Confidence Bounds on Trees

Upper Confidence bounds on Trees [27], or **UCT**, is a widely-used rollout-based **MCTS** value-estimation algorithm. **UCT** is an application of the bandit algorithm **Upper Confidence Bounds** [36], or **UCB**, to tree search. There are multiple variants to the details of how **UCT** can work, and since none of them relate directly to the work presented in this disseration, only one simple version of **UCT** will be discussed.

The **UTC** algorithm works by running a series of roll-outs through the state space, and making value estimates based on averages of observed returns.

The roll-outs follow trajectories generated by applying some exploration policy to the next-state sampler P. Once each trajectory has finished, the total discounted return experienced by each state-action pair is averaged with its previously observed returns.

The key to **UCT** is how it chooses its rollout policy. Variations exist, but the part they all have in common is the use of **UCB** for choosing actions in states that have been visited before.

The bandit algorithm **UCB** mixes exploration and exploitation. The algorithm will always choose the arm that has the greatest sum of average observed reward and a bonus term. The bonus term is a function of the number of times the arm in question has been pulled before, and how many chances the algorithm has had to pull an arm in total.

The bonus term used by **UCB** for arm i is $2C\sqrt{\frac{\log t}{n_i}}$, where t is the total number of pulls on any arm, n_i is the number of times arm i has been pulled, and C is a constant parameter that can be tuned. This bonus term has the nice properties of going to zero as t goes to infinity (allowing **UCB** to rely completely on average observed reward), and being infinite when n_i is zero (forcing **UCB** to try each action at least once).

UCT treats every possible state as a separate bandit algorithm, where each action represents a different bandit arm. As it visits a state again and again in its roll-outs, the **UCB** strategy will push it towards areas of the state-space that are either less explored or have had more favorable returns. This algorithm is also extremely computationally efficient, assuming next-states and rewards can be sampled quickly.

3.13 Forward Search Sparse Sampling

Forward Search Sparse Sampling [37], or **FSSS**, is another MCTS planner that preferentially expands the search tree through the use of rollouts. It is outlined in Algorithm 6. Unlike either **Bayesian Sparse Sampling** or **UCT**, it retains the attractive guarantees of the original **Sparse Sampling** algorithm. **FSSS** also maintains hard upper and lower bounds on the values for each state and action so as to direct the rollouts; actions are chosen greedily according to the upper bound on the value, and the next state is chosen such that it is the most uncertain of the available candidates (according to the difference in its upper and lower bounds).

FSSS will find the action to take from a given state s_0, the root of the search tree. The tree is expanded by running t trajectories, or rollouts, of length d. There are theoretically justified ways to choose t and d, but in practical applications they are knobs used to balance computational overhead and accuracy. To run a single rollout, the agent will call Algorithm 7, FSSS-Rollout$(s_0, d, 0, M)$. The values $U_d(s)$ and $L_d(s)$ are the upper and lower bounds on the value of the node for state s at depth d, respectively. Each time a rollout is performed, the tree will be expanded. After at most $(AC)^d$ rollouts are finished (but often less in practice), **FSSS** will have expanded the tree as much as is possibly useful, and will agree with the action chosen by **Sparse Sampling**. Thus, **FSSS** could be viewed as an anytime version of SS that uses pruning to speed its calculation.

The fact that **FSSS** maintains upper bounds on the values for each state it considers will be useful when an optimistic planner is required. It is often the case that accurately finding (or even approximating) a state's value will end up being very difficult or intractable. **FSSS** promises that, with high probability,

```
Input: state $s$, max depth $d$, #trajectories $t$, branching factor $C$, MDP $M$
Output: estimated value for state $s$
for $t$ times do
    FSSS-Rollout$(s, d, C, 0, M)$
$\hat{V}(s) \leftarrow U_d(s)$
return $\hat{V}(s)$
```

Algorithm 6: FSSS(s, d, C, t, M)

the true value for a given state will be somewhere in between $U(s)$ and $L(s)$. Maintaining this range of uncertainty can be useful for the algorithm employing **FSSS** as a planning algorithm, as we shall see in Chapter 5.

```
**Input**: state $s$, max depth $d$, branching factor $C$, current depth $l$, MDP $M$
if $Terminal(s)$ then
    $U_d(s) = L_d(s) = 0$
    return
if $d = l$ then
    return
if $\neg Visited_d(s)$ then
    $\text{Visited}_d(s) \leftarrow \text{true}$
    foreach $a \in A$ do
        $R_d(s,a), \text{Count}_d(s,a,s'), \text{Children}_d(s,a) \leftarrow 0, 0, \{\}$
        for $C$ times do
            $s', r \sim T_M(s,a), R_M(s,a)$
            $\text{Count}_d(s,a,s') \leftarrow \text{Count}_d(s,a,s') + 1$
            $\text{Children}_d(s,a) \leftarrow \text{Children}_d(s,a) \cup \{s'\}$
            $R_d(s,a) \leftarrow R_d(s,a) + r/C$
            if $\neg Visited_{d+1}(s')$ then
                $U_{d+1}(s'), L_{d+1}(s') \leftarrow V_{\max}, V_{\min}$
    $\text{Bellman-backup}(s,d)$
$a \leftarrow \text{argmax}_a U_d(s,a)$
$s' \leftarrow \text{argmax}_{s'} (U_{d+1}(s') - L_{d+1}(s')) \cdot \text{Count}_d(s,a,s')$
$\text{FSSS-Rollout}(s', d, l+1, M)$
$\text{Bellman-backup}(s,d)$
return
```

Algorithm 7: FSSS-Rollout(s, d, C, l, M)

```
**Input**: state $s$, depth $d$
foreach $a \in A$ do
    $U_d(s,a), L_d(s,a) \leftarrow 0, 0$
    foreach $s' \in Children_d(s,a)$ do
        $U_d(s,a) \leftarrow U_d(s,a) + \frac{1}{C}\gamma U_{d+1}(s')$
        $L_d(s,a) \leftarrow L_d(s,a) + \frac{1}{C}\gamma L_{d+1}(s')$
$U_d(s) = \text{argmax}_a U_d(s,a)$
$L_d(s) = \text{argmax}_a L_d(s,a)$
return
```

Algorithm 8: Bellman-backup(s, d)

Chapter 4
Best Of Sampled Set

Best Of Sampled Set [1], or **BOSS**, is a Bayesian approach to model-based reinforcement learning. This algorithm makes use of a general prior to sample entire models from the posterior. It works by sampling a set of models and combining them (see Figure 4.2) in a way that helps it choose the best in each situation.

This algorithm follows a strong tradition in model-based reinforcement learners of optimism in the face of uncertainty. One of the best-known model-based RL algorithms, **RMAX** [25], approaches this idea by locking unknown states to the highest possible value, or $V_{\max}$.

4.1 Algorithm

The **BOSS** algorithm, outlined in Figure 4.1, makes use of repeated samples from a posterior to create a hyper-model that can be used for trading off exploration and exploitation.

There are two distinct cases to which **BOSS** algorithm can be analyzed: in general and for discrete action and state spaces. The general case, for arbitrary state and action spaces, works as an effective exploration heuristic, while the specific case has formal efficiency guarantees. We will discuss first the general case, and then the discrete state-action case.

4.1.1 BOSS for General Models

The **BOSS** algorithm requires two important black boxes: a posterior sampler and a planner.

It is important to remember that these two black boxes encapsulate entire fields of active and interesting research and should not be thought of as solved problems when designing an agent. However, using **BOSS** as an exploration mechanism allows us to separate the reinforcement–learning problem from the planning and inference problems. This separation allows reinforcement–learning researchers to more easily draw on advances in other communities and to build on them.

The Posterior Sampler

The model posterior is the combination of a model prior (provided by the programmer) and observations (collected by the agent or provided by the environment). Techniques for deriving a posterior are covered in Chapter 2.

The posterior sampling black box must have the ability to sample from the set of worlds that are possible, given the observations made so far. The sampled model must be able to simulate transitions to next-states and rewards, given an action and the state it was taken from.

As the agent explores the world, taking actions and receiving feedback, it feeds the observations it has received to the posterior sampler. As the posterior sampler gets more and more data, the models sampled from it become more and more accurate, at least where the concentration of samples is high enough.

A posterior sampler can only be useful if some sort of accuracy guarantee is possible. That is, as more observations are made, models sampled from the posterior become closer to the truth, at least in the parts of the model associated with the observations. It is not always the case that a posterior will become more accurate with more data. A simple example is a prior that says that a coin will either always turn up heads or always turn up tails. The moment that both a head and tail have been observed, the posterior becomes broken. The posterior distribution for the coin's bias,

$$P(\rho|h=1,t=1) \quad = \quad \frac{\text{Bin}(H=1|\rho,n=2)P(\rho)}{P(H=1,n=2)}, \tag{4.1}$$

is now undefined. The denominator in Equation 4.1 is zero, since the coin must be either two-headed or two-tailed.

For the general case, I will not specify exact accuracy guarantees. For the special case in Section 4.1.2 there is a concrete assumption that must be met to guarantee efficiency.

The Planner

BOSS requires a planner that, given a model, can (probably) find an (approximately) optimal action for any given state. The specifics of how the planner comes to its decision, including the number of samples drawn from the model and how much time is spent in computation, are orthogonal to **BOSS**'s goal of low exploration complexity.

Fortunately, since the planner's operation is orthogonal to how **BOSS** works, the two can be analyzed separately.

Although the planner is decoupled from **BOSS**'s operation, it is tied closely to the posterior sampler. More accurately, it is closely tied to the models generated by the posterior sampler. The planner must be able to work in whatever model the posterior sampler can create. More than that, it must be able to work in a hyper-model, created by melding some number of the posterior sampler's models together, as discussed in Section 4.1.1.

Hyper-models

A hyper-model is the combination of some set of K models with identical action and state spaces, but possibly differing dynamics (as in Figure 4.3). The resulting hyper-model has the same state space as each of its sub-models, but has one more feature in its action space. This feature, $k \in \{1, ..., K\}$, indicates from which sub-model the dynamics of this action are taken. That is, the action $(2, a)$ has the same next-state and reward distributions that action a has in model 2.

Formally, a hypermodel m' can be constructed from k MDPs $m_1, m_2, ..., m_k$ that have the same state-space S and action-space A. They may each have unique transition and reward functions T and R. The hypermodel m' has state-space S and action-space $A \times \{i | i \in \mathbb{N}, i \geq 1, i \leq k\}$. The transition function for m' is defined to be

$$T_{m'}(s'|s, \langle a, i \rangle) \quad = \quad T_{m_i}(s'|s, a). \tag{4.2}$$

That is, the probability of ending up in state s' after taking the action $\langle a, i \rangle$ is the same as the probability of ending up in state s' when taking action a in MDP m_i. Similarly, the reward function for m' is defined to be

$$R_{m'}(s, \langle a, i \rangle) \quad = \quad R_{m_i}(s, a). \tag{4.3}$$

Intuitively, an agent acting according to m' is free to choose the dynamics from any of the sampled models during each step. An optimistic agent, one that thinks the best of the unknown, can choose the dynamics that give it the biggest advantage. This is an example of "optimism in the face of uncertainty", where the agent is drawn to parts of the state space that are either known to be good, or are not known. With this attack plan, the agent will either get good reward or get good knowledge. Since the amount of knowledge to be learned is usually finite, the agent will visit the unknown states enough to make them known, and be able to act without exploration in the future.

BOSS's method of model creation allows the hyper-model to retain all the optimism of any of its sub-models. The idea is that, for each state, at least one of the K models will predict dynamics at least as good as the true model's dynamics.

The claim that the hyper-model is, for every state, at least as optimistic as any of its sub-models is detailed in Lemma 3.

It is important to choose K carefully. A low value for K will increase the odds that, for some set of K models, there will be states for which *no* model's dynamics will be as good as the true model's — this situation breaks any guarantees that **BOSS** offers. A high value for K can cause two problems. First, the amount of work done by the planner increases, sometimes dramatically; the runtime complexity of most discrete-action planning algorithms relies linearly or super-linearly on the number of actions. Second, too much optimism can be a bad thing. In the extreme, as $K \to \infty$, all possible models will be represented in every set of K sub-models. In this situation, the agent's observations have no effect on the hyper-model, since the likelihood of each sampled model is not taken into account when compositing the sub-models together. With a finite, yet large, K, the number of observations required to have a particular state's dynamics be accurate in each of the sub-models grows as a polynomial of K. While this growth rate doesn't make a dent in any theoretical guarantees, it does present serious issues in practice.

Algorithm

The basic **BOSS** algorithm is as follows:

1. Sample K sub-models from the posterior.
2. Combine each of the K models sampled into a hyper-model, as in Section 4.1.1.
3. Using the planner, act optimally according to the hyper-model until a discovery event occurs.
4. Go back to Step 1.

This process is repeated forever, though eventually Step 3 will never complete, since the number of discovery events an agent can experience is bounded.

Here, we define a *discovery event* to be an action taken that causes a state to go from unknown to known. How this change is quantified varies with the model and prior being used. As an example, with a discrete state and action space, a state may become known once each action has been attempted from that state some pre-determined number of times. In a continuous state space, reaching some action density at a state could be defined to trigger a discovery event.

A discovery event is a signal to the agent that the observations it has gathered have affected the posterior sufficently that the K models it has been working with may now be out of date. A new hyper-model is created, from the new information, and the agent resumes its process of either getting high reward or making observations in states that are unknown (because they will have high reward in the hyper-model).

The agent will get high reward because states that are *known* to be good will be accurately represented in each of the K samples, and the planner will be using an accurate version of those states' dynamics when it plans, no matter which version of the action it chooses to use.

The agent will make observations in states that are *unknown* because, for each unknown state, at least one of the K sampled models will have dynamics that are optimistic for that state, making it more attractive to the agent than it would be otherwise.

4.1.2 BOSS for Discrete State and Action Spaces

With a discrete state and action space, where the model can be represented by an MDP, the **BOSS** algorithm can be defined more exactly, and given formal efficiency guarantees.

Knowness

For any prior that supports (has a non-zero probability for) the true model, there will be some minimum number of observed transitions from a particular state, with a particular action, that is large enough to guarantee (with high probability) a posterior sample that is close to the truth for that state-action pair. We will call this threshold B. We choose B such that after B next-states have been observed for a particular state-action pair, with high probability the true next-state distribution is close to the distribution inferred from the observation set. The threshold B is a function of the desired accuracy and probability of success.

A discovery event is then defined to occur on the B^{th} transition from a particular state-action pair.

4.2 Theoretical Guarantees

The **BOSS** algorithm is both PAC-MDP and PAC-BAMDP, under slightly different conditions. The difference between optimal and Bayes-optimal, in this situation, hinges on whether the "true" MDP has a bounded sample complexity or the prior itself is has a bounded sample complexity. This distinction is important to make, since with an especially strange prior, the posterior may not converge to something close to the true model after a reasonable number of observations. With this strange prior, **BOSS** will still be PAC-BAMDP, but the Bayes-optimal and optimal policies may differ significantly.

4.2.1 Definitions

This section formalizes some definitions that are important for understanding the theorem and proof that follows.

Definition 9. *An MDP $\hat{m}$ is said to be* locally ϵ-optimistic *at state-action pair s, a, with respect to a baseline MDP m_0, if*

$$R(s,a) + \gamma \sum_{s'} T_{\hat{m}}(s'|s,a) V_{m_0}(s') \quad \geq \quad R(s,a) + \gamma \sum_{s'} T_{m_0}(s'|s,a) V_{m_0}(s') - \epsilon,$$

or

$$\sum_{s'} T_{\hat{m}}(s'|s,a) V_{m_0}(s') \quad \geq \quad \sum_{s'} T_{m_0}(s'|s,a) V_{m_0}(s') - \epsilon, \tag{4.4}$$

where R is the reward function for MDP m, where T_m is the transition function for MDP m, $V_m(s)$ is the value of state s in MDP m.

That is, if the agent is able to take a single step from s, a in $\hat{m}$, and then must exist in m_0 after that, it is able to achieve an expected discounted return that is at least as high as that that it could achieve starting in m_0, less ϵ.

Definition 10. *An MDP $\hat{m}$ is said to be* locally ϵ-accurate *at state-action pair s, a, with respect to a baseline MDP m_0, if*

$$\|T_{\hat{m}}(\cdot|s,a) - T_{m_0}(\cdot|s,a)\|_\infty \quad \leq \quad \epsilon,$$

or

$$\forall_{s'} \left|T_{\hat{m}}(s'|s,a) - T_{m_0}(s'|s,a)\right| \quad \leq \quad \epsilon, \tag{4.5}$$

where T_m is the transition function for MDP m.

That is, the likelihood of ending up in state s' while in $\hat{m}$ is ϵ-close to that likelihood in m.

With high probability, when the prior is has a sample complexity of B, posterior samples will be locally ϵ-accurate for state-action pairs with at least B observed transitions.

Definition 11. *A state-action pair that has been tried at least B times is declared to be* known. *If a state-action pair has been tried fewer than B times, it is declared to be* unknown.

If there is any action a for which the state-action pair s, a is unknown, *the state s is also* unknown. *A state is* known *if all state-action pairs including it are* known.

4.2.2 BOSS is PAC-BAMDP

Theorem 1. *The **BOSS** algorithm, when provided with*

1. *a prior MDP distribution ϕ,*

2. *a knownness threshold B,*

3. *a samples-per-hyper-model count K,*

and if

1. *the prior ϕ is has a sample complexity of B, with ϵ_0 andδ_0,*

2. *a posterior sample is* locally ϵ_0-optimistic *for any arbitrary state-action pair with probability at least $1 - \delta_1$,*

where $\epsilon_0 = \epsilon(1-\gamma)/(\gamma V_{\max})$, $\delta_0 = \delta/(2KS^2A^2)$, and $\delta_1 = \left(\delta/(2S^2A^2)\right)^{1/K}$, will, with probability at least $1-\delta$, perform ϵ-Bayes-optimal actions for all but a number of steps that is polynomial with ϵ, δ, γ, B, the number of states S, and the number of actions A. That is, ***BOSS*** *will be PAC-BAMDP.*

4.3 Proof of PAC-BAMDP

In this section, I present a proof that **BOSS** is PAC-BAMDP in discrete state and action spaces, as long as certain assumptions hold. That is, with probability $1-\delta$, the agent will choose ϵ-Bayes-optimal actions for all but a number of steps that is polynomial with the number of states S, the number of actions A, the prior's accuracy threshold B, ϵ and δ.

This proof depends on three main assumptions, which follow.

1. Bounded discoveries: the agent will visit *unknown* state-action pairs only a small number of times.

2. Optimism: the values for each *unknown* state in every hyper-model are optimistic.

3. Accuracy: the hyper-models have accurate dynamics for any state-action pair that is *known*.

These three assumptions are sufficient to ensure, with high probability, a number of sub-optimal steps in the BAMDP that is polynomial [10].

We declare a state-action pair to be *known* if it has been attempted at least B times. This number of attempts allows the agent to have gathered enough experience to be able to accurately simulate its dynamics, according to Assumption 3. If the state-action pair has been attempted fewer than B times, not enough observations have been made to make an accurate prediction, and it is considered *unknown*. *Unknown* states have optimistic values in the hyper-models, incentivizing the agent to visit them and learn about their dynamics.

In particular, because the prior ϕ must have bounded sample complexity, we can assume that the dynamics for that state-action pair in a posterior sample are ϵ_0-accurate, for $\epsilon_0 = \epsilon(1-\gamma)/(\gamma V_{\max})$, with high probability. If all state-action pairs in a sampled MDP have ϵ_0-accurate dynamics with respect to the "true" MDP, then the simulation lemma [12, 38] tells us that the optimal policy for that sampled MDP will be an ϵ-optimal policy in the "true" MDP.

4.3.1 Bounded Discoveries

Since a new hyper-model is constructed each time a discovery event occurs, it is important to bound the number of possible discovery events. In the discrete state and discrete action case, a discovery event occurs when a particular state-action pair is tried for the B^{th} time. Since the number of unique state-action pairs is SA, there may only be SA discovery events. As a result, the maximum number of hyper-models created over the agent's lifetime is SA.

4.3.2 Optimism

This proof depends on hyper-models being ϵ-optimistic for all *unknown* state-action pairs. Intuitively, if *unknown* state-action pairs are optimistic, the agent has incentive to visit them and try them, eventually causing them to become *known*.

Condition 2 in Section 4.2.2 lets us know that for an arbitrary *unknown* state-action pair, an MDP $\hat{m} \sim \phi|h$ sampled from the posterior will be *locally* ϵ_0-*optimistic* for that pair with probability at least $1 - \delta_1$.

To have an optimistic hyper-model, it is sufficient that for each of the SA state-action pairs s, a, at least one of the K samples is *locally* ϵ_0-*optimistic* at s, a. Alternatively, that each state s has a *locally* ϵ_0-*optimistic* version of each action.

First we show that the construction of a hyper-model retains all the optimism of its sub-models.

Lemma 3. *A hyper-model m' has a value for each state at least as high as the value of that state in each of the sub-models $m_1, m_2, ..., m_k$.*

Proof. By augmenting the set of actions available for a single state, we can only raise the value of that state since the value, defined by the Bellman equation,

$$V_m^*(s) \quad = \quad \max_{a \in A} R_m(s,a) + \gamma \sum_{s'} T_m(s'|s,a) V_m^*(s'), \tag{4.6}$$

is taken from the max over the set of actions. Augmenting the set of actions for s_0 results in a slightly different recurrence relation,

$$V_m^*(s_0) \quad = \quad \max_{a \in A \bigcup A'} R_m(s_0,a) + \gamma \sum_{s'} T_m(s'|s_0,a) V_m^*(s'), \tag{4.7}$$

$$V_m^*(s \neq s_0) \quad = \quad \max_{a \in A} R_m(s,a) + \gamma \sum_{s'} T_m(s'|s,a) V_m^*(s'). \tag{4.8}$$

Since $V_m^*(s_0)$ can only have increased, we know the values of every other state can also only have increased, since $T_m(s'|s,a) \geq 0$ is a non-negative probability.

It follows that as we augment the actions available to each state individually, the values of any state in the new MDP must be at least as great as they were before those actions were added. Because m' is an augmentation for each of the sub-models m_i, it follows that $\forall_{i,s} V_{m'}(s) \geq V_{m_i}(s)$. □

Given that we know hyper-models retain all the optimism of their sub-models, we can bound the likelihood that a hyper-model is optimistic everywhere.

Lemma 4. *For any given* unknown *state-action pair, if a model sampled from the posterior is locally optimistic with probability at least* $1 - \delta_1$*, then a hyper-model created from the combination of K models sampled independently from the posterior is optimistic for every* unknown *state-action pair with probability at least* $1 - SA{\delta_1}^K$.

Proof. For a specific *unknown* state-action pair s, a, the probability that one of the K samples used to construct a single hyper-model is *locally* ϵ_0*-optimistic* is the complement of the probability that none of them are.

The probability that a single sample is not locally ϵ_0-optimistic is at most δ_1.

Since these K samples are conditioned on the same data and are independent with respect to that data, the probability of K samples not being locally ϵ_0-optimistic is at most ${\delta_1}^K$.

The probability that at least one of them is locally ϵ_0-optimistic is then no less than the compliment, $1 - {\delta_1}^K$.

These K sub-models are composed into a single hyper-model, which needs to be locally ϵ_0-optimistic for all *unknown* state-action pairs. At any given point, there are no more than SA *unknown* state-action pairs. The union bound tells us that the likelihood of this being the case for a particular hyper-model is at least $1 - SA{\delta_1}^K$. □

Since the number of hyper-models created over the lifetime of the agent is bounded, so is the likelihood that they are all optimistic.

Lemma 5. *If a posterior sample is* locally ϵ_0-optimistic *for any given* unknown *state-action pair with probability at least* $1 - \delta_1$*, then all* unknown *states in all hyper-models will have* locally ϵ_0-optimistic *version of each action with probability at least* $1 - \delta/2$.

Proof. Since an agent may experience at most SA discovery events and, therefore, construct at most SA hyper-models, and the likelihood that a particular hyper-model is optimistic is at least $1 - SA{\delta_1}^K$, the union bound tells us that the likelihood of each of them being locally ϵ_0-optimistic for all of their *unknown* state-action pairs is at least $1 - S^2A^2{\delta_1}^K$.

Since $\delta_1 = \left(\delta/(2S^2A^2)\right)^{1/K}$, all hyper-models will be locally ϵ_0-optimistic for all of their *unknown* state-action pairs with probability at least $1 - \delta/2$. □

4.3.3 Accuracy

The **BOSS** algorithm, by itself, cannot guarantee model accuracy; as a Bayesian algorithm, it is dependent on the prior chosen for the agent.

Lemma 6. *If the prior ϕ is has a sample complexity of B for δ_0 and ϵ_0, then every* known *state-action pair in every hyper-model will be* locally ϵ_0-accurate *with probability at least* $1 - \delta/2$.

Proof. Since ϕ has bounded sample complexity, for a given posterior sample, a particular *known* state-action pair will be *locally ϵ_0-accurate* with probability at least $1 - \delta_0$.

Since the number of constructed hyper-models is at most SA, and K posterior samples go into each hyper-model, *known* state-action pairs have to be *locally ϵ_0-accurate* in at most KSA posterior samples. Each time the posterior is sampled, there are at most SA *known* state-action pairs, so the number of times a *known* state-action pair needs a *locally ϵ_0-accurate* is at most KS^2A^2.

Since $\delta_0 = \delta/(2KS^2A^2)$, applying the union bound tells us that the likelihood that all *known* state-action pairs in all posterior samples are *locally ϵ_0-accurate* is at least $1 - KS^2A^2\delta_0 = 1 - \delta/2$.

□

4.3.4 Probability of Success

Now that optimism and accuracy are guaranteed with high probability, they can also be jointly guaranteed.

Lemma 7. *In every posterior sample, all* unknown *state-action pairs are* locallaly ϵ_0-optimistic*, and all* unknown *state-action pairs are* locallaly ϵ_0-accurate*, with probability at least* $1 - \delta$*.*

Proof. Lemma 5 shows that all *unknown* state-action pairs are *locallaly ϵ_0-optimistic* in all samples with probability at least $1 - \delta/2$.

Lemma 6 shows that all *known* state-action pairs are *locallaly ϵ_0-accurate* in all samples with probability at least $1 - \delta/2$.

Applying the union bound to these two events holds both true with probability at least $1 - \delta$. □

4.3.5 Approximately Bayes-optimal with High Probability

Lemma 7 shows that all *known* state-action pairs will be *locally ϵ_0-accurate* and all *unknown* state-action pairs will be *locally ϵ_0-optimistic*, in all hyper-models over the course of the agent's lifetime, with high probability. These facts, along with the bounded number of discoveries possible, allow us to connect **BOSS** to the general PAC-MDP Theorem, detailed in Section 1.1.4, using the BAMDP.

4.4 Proof of PAC-MDP

By strengthening one of the conditions, we can ensure that **BOSS** is also PAC-MDP.

Condition 1 in Section 4.3 stated that the prior ϕ must have a sample complexity of B. That is, if the "true" model m_0 is sampled from ϕ, and some observations are made from acting in m_0, then with high probability a posterior sample will have accurate dynamics for all state-action pairs with enough example transitions.

To create a PAC-MDP version, we replace that condition with a new one:

1. The "true" model, m_0, has a sample complexity of B with the prior ϕ.

After this, the proof is identical. The distinction lies in that m_0 is no longer necessarily tied to the prior in any way — it is only important that the prior can be used to *learn* m_0.

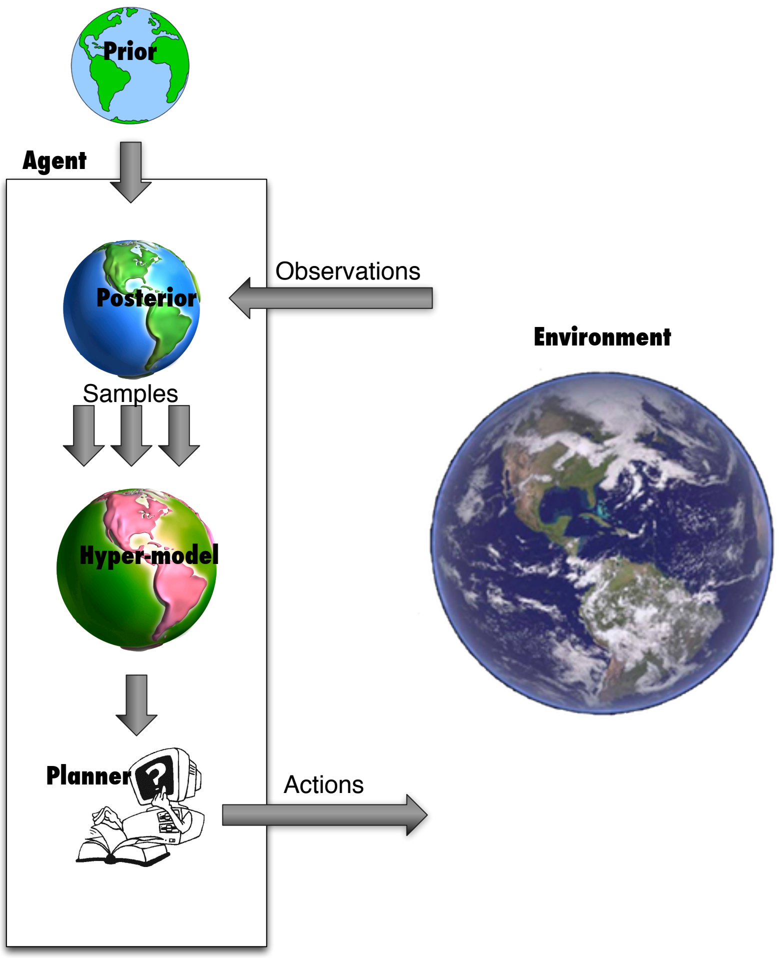

Figure 4.1: **BOSS** combines observations made from the environment with the prior to create sets of posterior samples. These posterior samples are combined to create a hyper-model, which the planner takes as its model. The policy on the hyper-model is used in the environment.

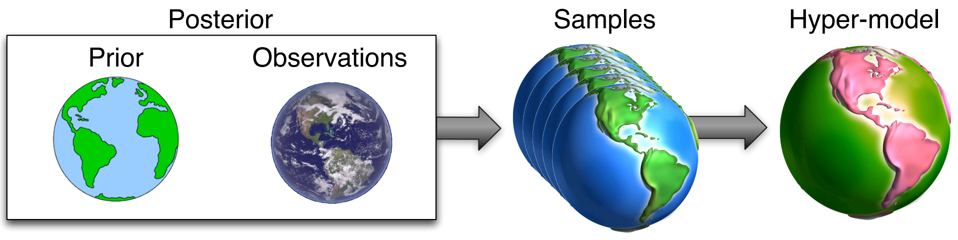

Figure 4.2: A posterior is created by combining the prior with observations made by the agent in the environment. Every the agent decides that it has learned something, several samples are drawn from the posterior and combined to create the hyper-model.

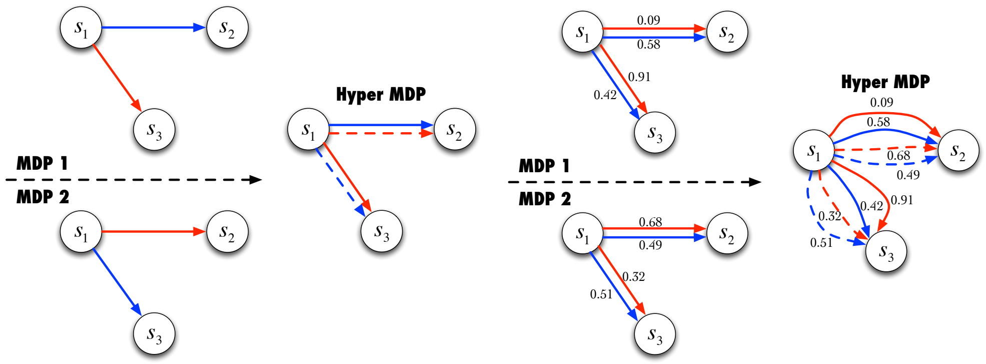

Figure 4.3: **Left:** Two samples with the same state- and action-spaces and deterministic dynamics are combined to create a hyper-model. Every action from each of the provided models exists in the hyper-model. **Right:** Two stochastic MDPs are combined to create a stochastic hyper-model MDP. In both cases, solid lines in the hyper-model indicate the action was taken from the upper sample, and dashed lines indicate the action was taken from the lower sample.

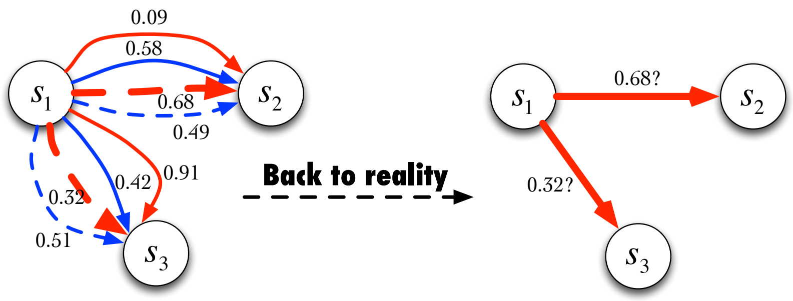

Figure 4.4: The agent chooses the best policy in the hyper-model, and then translates it back into the environment it is dealing with. In this case, it chose the red dashed action (perhaps s_2 has a higher value than s_3. In the real environment, the agent will choose the red action, hoping that it has a high probability of resulting in transition to s_3. If this hope is in error, the agent may try the action several times and eventually have enough evidence to effect a change in the posterior.

Chapter 5
Bayesian Forward Search Sparse Sampling

Bayesian Forward Search Sparse Sampling, or **BFS3**, is a method for applying a special kind of tree search to a Bayes-Adaptive Markov Decision Process (BAMDP). **BFS3**, introduced in Section 5.2 of this Chapter, uses **Forward Search Sparse Sampling** as a subroutine. This application results in a policy with provably efficient sample complexity.

5.1 Building Blocks

BFS3 is built upon the ideas from a few existing algorithms.

5.1.1 Sparse Sampling

Sparse Sampling [39] works by recursively expanding a full search tree up to a certain depth d. At the root, each of the A actions is used for sampling a constant number of times C, yielding a set of $A \cdot C$ children. Sparse sampling is then run on each of the children with a recursion depth one less than the root's. Once the tree is fully created, the leaves are each assigned a value of zero. Then, starting at the leaves, the values are backed up and combined via the Bellman equation, defining the parents' values, until the root's value is determined. The total number of nodes visited in this search tree is $(AC)^d$, making the algorithm impractical to run in all but the most trivial of domains.

It is worth noting, however, that **Sparse Sampling** is best known as one of the first reinforcement-learning planning algorithms that can achieve high accuracy with high probability using an amount of computation that is not a function of the size of the state space[1]. Because of this attractive property, it makes sense to select it or one of its variants as the planner for the infinitely large BAMDP. **Sparse Sampling** is the basis for a number of Monte-Carlo Tree Search (MCTS) algorithms, which are considerably faster in practice [18, 27, 37].

Sparse Sampling is discussed in more detail in Chapter 3.

5.1.2 Forward Search Sparse Sampling

Forward Search Sparse Sampling, or **FSSS**, is a Monte-carlo tree search algorithm used for planning in MDPs. It preferentially expands the search tree through the use of rollouts, and is outlined in Algorithm 6. Unlike either **Bayesian Sparse Sampling** [18] or **UCT** [27], it retains the attractive guarantees of the original **Sparse Sampling** algorithm. An important property of **FSSS** is that it maintains hard upper and lower bounds, with high probability, on the values for each state and action, and uses those bounds to direct the rollouts; actions are chosen greedily according to the upper bound on the value, and the next state is chosen such that it is the most uncertain of the available candidates (according to the difference in its upper and lower bounds).

The values $U_d(s)$ and $L_d(s)$ are the upper and lower bounds on the value of the node for state s at depth d, respectively. Each time a rollout is performed,

[1] This lack of dependence on the number of states assumes that sampling from the model can be done in constant time. In most real situations there is at least a logarithmic dependency on the number of states just for representing any given state.

the tree will be expanded. After at most $(AC)^d$ rollouts are finished (but often less in practice), **FSSS** will have expanded the tree as much as is possibly useful, and will agree with the action chosen by **Sparse Sampling**.

FSSS is discussed in more detail in Chapter 3.

5.1.3 A Modification on FSSS

BFS3 uses a slightly modified version of **FSSS**.

1. **FSSS** will not keep different values based on search depth. That is, $\forall_{d,d'} U_d = U_{d'}$, $L_d = L_{d'}$, $R_d = R_{d'}$, $\text{Children}_d = \text{Children}_{d'}$, $\text{Visited}_d = \text{Visited}_{d'}$, and $\text{Count}_d = \text{Count}_{d'}$. From this point forward, the depth suffix will be dropped.

2. Since **FSSS** is executed at time-step t, call its upper and lower bound functions U_t and L_t. Also, have time-step specific versions of each of the other functions (R, Children, Visited, and Count). Before **FSSS** is invoked by **BFS3**, let $U_t = U_{t-1}$, $L_t = L_{t-1}$, $R_t = R_{t-1}$, $\text{Children}_t = \text{Children}_{t-1}$, $\text{Visited}_t = \text{Visited}_{t-1}$, and $\text{Count}_t = \text{Count}_{t-1}$.

The practical effect these changes have is that there is now a single MDP, which we will call $m_{\mathcal{A}}$, that every call to **FSSS** uses for making its value estimates, and that from time step to time step its estimates will build upon the estimates of the past. **FSSS**'s upper and lower bounds for the value of states in $m_{\mathcal{A}}$ are true, not probabily approximate, since **FSSS** is using the true transition function for this MDP.

```
**Input**: time step $t$, state $s$, history $h$, depth $d$, branching factor $C$,
        #trajectories $T$, MDP prior $\phi$
**Output**: action to take in state $s$
Let $x = \langle s, h \rangle$
**if** $x \in K$ **then**
    **return** $\pi_t(x)$
**foreach** $a \in A$ **do**
    **if** $Children_t(x) = \{\}$ **then**
        $x', r \sim T\text{-}R_\phi(x, a)$
        $\text{Children}_t(x) \leftarrow \text{Children}_t(x) \bigcup \{x'\}$
        $\text{Count}_t(x, a, x') \leftarrow \text{Count}_t(x, a, x') + 1$
        $R_t(x, a) \leftarrow R_t(x, a) + \frac{r}{C}$
    **foreach** $x' \in Children_t(x, a)$ **do**
        Run $FSSS(x', d, C, T, M_\phi)$
    $U_t(x, a) \leftarrow R(x, a)$
    **foreach** $x' \in Children_t(x, a)$ **do**
        $U_t(x, a) \leftarrow U_t(x, a) + \gamma \frac{\text{Count}_{t(x,a,x')}}{C} U_t(x')$
**if** $\pi_t \neq \pi_{t-1}$ **then**
    $K \leftarrow \{\}$
$K \leftarrow K \bigcup \{x\}$
**return** $\pi_t(x)$
```

Algorithm 9: BFS3(t, s, h, d, C, T, ϕ)

5.2 Bayesian Forward Search Sparse Sampling

BFS3 is the application of **FSSS** to a Bayes-adaptive MDP, or BAMDP. The BAMDP is defined by the MDP prior $\phi(M)$, and the joint transition and reward function $T\text{-}R_\phi$ is constructed such that

$$P(\langle s', h + (s, a, s', r) \rangle, r | \langle s, h \rangle, a) \quad = \quad \int_M P(s', r | s, a, M) \phi(M | h) dM.$$

Here, the BAMDP's state-space is the set of belief-states that include the history of all transitions seen so far. Because of how the BAMDP's transition function is constructed, each possible next-state's history includes the transition from the previous state to the next state.

BAMDPs, their construction, and their relation to Bayes-optimality are discussed in detail in Section 1.1.4.

Since, with **FSSS**, the next belief-states are only sampled and their likelihoods are never calculated, a simple generative process can be used:

$$M \sim \phi|h \tag{5.1}$$

$$s', r \sim T_M(s,a), R_M(s,a). \tag{5.2}$$

This process is used whenever **BFS3** or its subroutine **FSSS** sample a next-state and reward. The algorithm never holds on to an individual MDP after a single transition has been sampled from it. Also, note that whenever **FSSS** does a Bellman backup, that backup is done for a belief-state (since **FSSS** is acting on the BAMDP).

First, an MDP M is sampled from the posterior $\phi|h$. Once it is sampled, then the next state and reward are sampled from M. Sometimes this posterior sampling can be computationally expensive, since inference is generally a hard probelm. The **BFS3** algorithm can only work effectively if this step is fast. Fast posterior sampling is possible with many priors, including the very simple FDM, but also more structured priors can be used with efficient inference — all the priors used with **BFS3** in Chapter 6 have efficient posterior inference.

To reconstruct the resulting belief-state, we pack the resulting concrete state s' with the new history made by augmenting h with (s, a, s', r), resulting in a transition from belief-state $\langle s, h\rangle$ to $\langle s', h + (s, a, s', r)\rangle$, except in cases where h has B examples of transitions from (s, a), where the new belief state, $\langle s', h\rangle$, has an unchanged history. In many cases, the history h can be summarized by a more compact sufficient statistic. For instance, with discrete-state and -action MDPs, only a histogram of next-states needs to be kept for each state-action pair, and the exact sequence that these transitions happened in is irrelevant.

Figure 5.1 illustrates **BFS3**'s process for each belief-state visited by the agent.

In future steps through the environment, the agent may find itself in one of the reachable belief-states in the search tree — all of the reachable belief-states represent one possible concrete state and history that, with some probability, may be the root of **BFS3**'s search tree in some future situation.

5.2.1 Choice of FSSS

Although it is meaningful to use any sample-based planner for **BFS3**, I choose **FSSS** because of a particular promise that it makes: with high probability, **FSSS** will never underestimate the value of a state, even if it is unable to estimate it accurately. Other sample-based planners well-known in the reinforcement-learning literature, such as **UCT** [27], do not share this guarantee and it is important for the guarantees made by **BFS3** in Section 5.2.3. Currently, **FSSS** is the only known algorithm that can satisfy the conditions in Section 5.2.3.

5.2.2 Algorithm

The **BFS3** process is detailed in Algorithm 9, which references Algorithm 6 in Chapter 3. Every time the agent arrives in a new belief-state $\langle s, h \rangle$, it simulates each action C times, to get a set of $C \times A$ next belief-states and rewards. Then, **FSSS** is used to find the upper bound on the value of each of the next belief-states. Once their optimistic values are found, **BFS3** will average all the sums of the discounted values and rewards together to get an estimate of the Bayes-expected value of taking that action, and choose the action that maximizes this quantity.

All histories should only take into account the first B transitions from a given state-action pair, where B is the number of transitions required to have

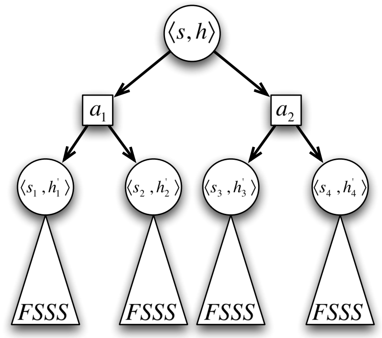
$\langle s, h \rangle$
a_1
a_2
$\langle s_1, h_1' \rangle$
$\langle s_2, h_2' \rangle$
$\langle s_3, h_3' \rangle$
$\langle s_4, h_4' \rangle$
FSSS
FSSS
FSSS
FSSS

Figure 5.1: **BFS3** samples next-belief-states for each action C times, and then runs **FSSS** on the resulting belief state, using the BAMDP as a generative model. Every node that is the same distance from the root represents one of the possible worlds that the agent may experience, each with a different history and MDP posterior.

accurate posterior samples for any particular state-action. The exact value for B is determined by the prior's sample complexity, and is tied to the accuracy condition in Section 5.2.3, Condition 1.

Putting a limit on the number of transitions to remember for a particular state-action pair does two important things. First, it means it is possible for the agent to end up in the same effective belief-state more than once[2]. The result is that **BFS3** can reuse past decisions and puts a bound on the number of decisions that can be made, saving computation time. Second, it aids in the PAC-BAMDP proof, since the number of random events that need to succeed is bounded.

[2] Normally, an agent would never experience the same belief-state twice, because every step it takes grows the history by one transition.

5.2.3 BFS3 is PAC-BAMDP

If certain reasonable prerequisites are satisfied, then with high probability, **BFS3** will choose actions that are approximately Bayes-optimal except for a small number of times.

Definition 12. *Let $m_{\mathcal{A}}$ be the MDP given to **FSSS**, as described in Section 5.1.3, and let $\pi_t(x) = \text{argmax}_a Q_t(x, a)$, where Q_t is the upper bound on the value, as determined by **FSSS** at time-step t.*

Definition 13. *An* unknown *state-action pair is one with fewer than B visits. A* known *state-action pair is any state-action pair with at least B visits.*

Definition 14. *A* pinned *belief state is one with an ϵ_A-accurate value according to V_t. That is, $|V_t(s) - V_{\mathcal{A}}(s)| \leq \epsilon_A$. It is important to note that the agent does not necessarily know which belief states are pinned.*

Definition 15. *Let C be the quantity such that C samples from a transition function are enough to create an ϵ_T^C-accurate approximation with probability at least $1 - \delta_T^C$, and such that C samples from a reward function are enough to create an ϵ_R^C-accurate approximation with probability at least $1 - \delta_R^C$. Since **BFS3**'s sample complexity does not depend on a particular value for C, only on the accuracies and success probabilities, it is sufficient to assert that such a value for C exists, without finding it, for the proof of PAC-BAMDP.*

Theorem 2. *With probability at least $1 - \delta$, the expected number of sub-ϵ-Bayes-optimal actions taken by **BFS3** is at most $BSA(S + 1)d/\delta_l$ if the following assumptions are true.*

1. *Accuracy: The prior's transition function and reward function have a sample complexity of at most B, for a transition accuracy of ϵ_T with probability at least $1 - \delta_T$, and a reward accuracy of ϵ_R with probability at least $1 - \delta_R$.*

2. *Optimism: If running **FSSS** on belief-state x results in $V_t(x) > V_A(x) + \epsilon_A$ (that is, afterwards x remains an unpinned belief state), then π_t leads from x through a sequence of at most d unpinned belief states, ending with an unknown state-action pair, with probability at least δ_l, where d is the search depth used by **FSSS**.*

Proof Sketch:

First, we will show that there is a BAMDP, constructed from the prior ϕ, whose optimal policy is the Bayes-optimal policy for $m_0 \sim \phi$. Then, we will show that **BFS3**, acting in the BAMDP, will satisfy the three criteria required for PAC-MDP behavior [10, 40] in that BAMDP[3]. These criteria are: 1. accuracy, 2. bounded discoveries, and 3. optimism.

First, because of Assumption 1 in our theorem statement, we know that once we have received B examples of transitions from a state-action pair (s, a), our estimate of the next-state distribution for that pair will be accurate. (This condition need not hold for degenerate priors, but it appears to hold quite broadly.)

Second, since we forget all additional transitions from state-action pairs for which we have seen B examples, the number of possible state-histories that an agent can observe is bounded. Specifically, each time a transition from some state-action (s, a) is observed, either no change will be made to the state-action's histogram (it already sums to B), or exactly one entry in the histogram will be incremented by 1. Since the histogram can be changed at most B times, the total number of histories possible for an agent over the course of a single experiment is $B \cdot S \cdot A$ (B histories for each state-action pair).

[3] PAC-MDP behavior in the BAMDP implies near Bayes-optimal behavior in the learning setting, as discussed in Section 3.2.

A discovery event, or one that potentially changes the MDP posterior, is an event that results in a change to the history. There are $B \cdot S \cdot A$ discoveries possible, since other transitions will be forgotten.

Third, **FSSS**(x', d, C, T, M_ϕ) is guaranteed to have an optimistic value estimate for belief-state x' as T (the number of trajectories), our bounded resource, grows smaller. We also know that, from Assumption 2 of the theorem, T is sufficient to find accurate estimates of x' if all states in s''s subtree have converged next-state posteriors. Simply put, if x''s subtree has no unknown state-action pairs, then **FSSS**'s estimate of that state's value will be accurate. As a result, if **FSSS**'s estimate of a state's value is inaccurate, there must be something to learn about in x''s subtree. **FSSS** guarantees that this inaccuracy will be optimistic.

Also possible is that the value estimate of x' is accurate *and* there are unknown states in its subtree. In this case, the agent can decide whether or not to visit that state fully informed of its value, and can take a Bayes-optimal action.

The PAC-MDP criteria direct the agent to areas of either high value or high uncertainty, managing the exploration/exploitation tradeoff. Because the agent will only go to areas of high uncertainty over areas of high reward a bounded number of times that grows linearly with the number of possible discovery events, we bound the number of sub-optimal steps taken over the lifetime of the agent. □

5.2.4 Proof of PAC-BAMDP

This section presents the detailed argument that **BFS3** is near Bayes-optimal.

There are two distinct steps to this proof. The first step is to find the likelihood that all estimates are accurate. That likelihood is then the likelihood that

BFS3 will be ϵ-Bayes optimal for all but a small expected number of steps. The second step is to bound that number, assuming all estimates are accurate.

Definitions

This section will list several definitions important to the proof.

Definition 16. *Let H be the set of all histories, which are sequences of (s, a, r, s') transitions.*

Definition 17. *Let $X = S \times H$ be the set of belief states.*

Definition 18. *Let the BAMDP m_ϕ be the MDP with state space X, action space A, and the transition and reward functions:*

$$\begin{aligned} T_{m_\phi}(x'|x,a) &= T_{m_\phi}(\langle s',h'\rangle|\langle s,h\rangle\rangle,a), && (5.3)\\ T_{m_\phi}(\langle s',h'\rangle|\langle s,h\rangle\rangle,a) &= f(h'|h,s,a)\int_m \phi(m|h)T_m(s'|s,a)dm, && (5.4)\\ f(h'|h,s,a) &= \mathbb{1}[h' = h + (s,a,r,s')], && (5.5)\\ R_{m_\phi}(x,a) = R_{m_\phi}(\langle s,h\rangle\rangle,a) &= \int_m \phi(m|h)R_m(s,a). && (5.6) \end{aligned}$$

That is, m_ϕ is the BAMDP based on the prior ϕ.

Definition 19. *Let $n_h(s,a)$ be the number of transitions from (s,a) in h.*

Definition 20. *Let the set of possible histories $H^B \subset H$ be the set of histories with no more than B examples of transitions from any state-action pair. That is,*

$$H^B = \{h | h \in H \wedge \forall_{s,a} c_h(s,a) \leq B\}. \quad (5.7)$$

Definition 21. *Let the BAMDP m_ϕ^B be the MDP with state space $X^B = S \times H^B$,* *action space A, and the transition and reward functions:*

$$\begin{aligned} T_{m_\phi^B}(x'|x,a) = T_{m_\phi}(\langle s',h'\rangle|\langle s,h\rangle\rangle,a) &= f_B(h'|h,s,a)\int_m \phi(m|h)T_m(s'|s,a)dm, && (5.8)\\ f_B(h'|h,s,a) &= \begin{cases} n_h(s,a) \leq B & : \mathbb{1}[h' = h + (s,a,r,s')] \\ n_h(s,a) = B & : \mathbb{1}[h' = h], \end{cases} && (5.9)\\ R_{m_\phi^B}(x,a) &= R_{m_\phi}(x,a). && (5.10) \end{aligned}$$

That is, m_ϕ^B is the BAMDP based on the prior ϕ, where transitions from state-action pairs with at least B examples in the history are forgotten. Since the transition function has a sample complexity of B, with an accuracy of ϵ_T, and the reward function has a sample complexity of B with an accuracy of ϵ_R, m_ϕ^B is an ϵ_T, ϵ_R approximation of m_ϕ.

Definition 22. *Let the BAMDP $m_\mathcal{A}$ be the MDP with state space $X^B = S \times H^B$, action space A, and the transition and reward functions $T_\mathcal{A}$ and $R_\mathcal{A}$, where*

$$\begin{aligned} \forall_{x',x,a}|T_\mathcal{A}(x'|x,a) - T_{m_\phi^B}(x'|x,a)| &\leq \epsilon_T^c, && (5.11)\\ \forall_{x,a}|R_\mathcal{A}(x,a) - R_{m_\phi^B}(x,a)| &\leq \epsilon_R^c. && (5.12) \end{aligned}$$

That is, $m_\mathcal{A}$ is an ϵ_T^c, ϵ_R^c approximation of m_ϕ^B.

Accuracy

This section will find the likelihood that all estimates are accurate.

There are two sets of estimates to be made. The first set is the accuracy of a transition function estimate and reward function, given that estimate, given C samples from the functions being estimated. We will call such an estimate a C-estimate.

Lemma 8. *All C-estimates of the transition and reward function of m_ϕ^B will be ϵ_T^c-accurate and ϵ_R^c-accurate with probability at least $1 - \delta_C$, where*

$$\delta_C = BS^2A^2(\delta_T^C + \delta_R^C). \quad (5.13)$$

Proof. The probability that, for a given history, state, and action, the transition and reward estimates both fall within their bounds, is at least $1 - (\delta_T^C + \delta_R^C)$, by the union bound.

Since the maximum size of a history is BSA (since there may be at most B examples of each state-action pair), and since the history used by **BFS3** will never shrink, the agent can experience at most BSA histories over the course of its lifetime. For each of those histories, there are at most SA state-action pairs for which it may make estimates. Multiplying those two together limits the number of history-state-action combinations at BS^2A^2.

The probability that all C-estimates for all state-action-history combinations are correct, over the lifetime of the agent, is $1 - BS^2A^2(\delta_T^C + \delta_R^C)$ by the union bound. □

The second set is the estimate of the posterior after B examples are observed from the MDP drawn from the prior. We will call such an estimate a B-estimate.

Lemma 9. *All B-estimates of the transition and reward function of m_ϕ will be ϵ_T-accurate and ϵ_R-accurate with probability at least $1 - \delta_B$, where*

$$\delta_B \quad = \quad SA(\delta_T + \delta_R). \tag{5.14}$$

Proof. The probability that, for a given state-action pair with B examples, the posterior transition and reward functions both fall within their bounds is at least $1 - (\delta_T + \delta_R)$, by the union bound.

The maximum number of state-action pairs that can ever have B examples is SA (all of them).

The probability that all B-estimates for all state-action pairs are correct, over the lifetime of the agent, is $1 - SA(\delta_T + \delta_R)$ by the union bound. □

The likelihood that all estimates of any sort are accurate needs to be established.

Lemma 10. *All C-estimates and B-estimates made by* ***BFS3*** *fall within their bounds with probability at least* $1 - \delta$*, where*

$$\delta \quad = \quad BS^2A^2(\delta_T^C + \delta_R^C) + SA(\delta_T + \delta_R). \tag{5.15}$$

Proof. Proof follows immediately from the union bound. □

Lemma 11. *With probability at least* $1 - \delta$*,* $m_{\mathcal{A}}$ *is an* ϵ_T^C*,*ϵ_R^C*-approximation of* m_ϕ^B*, and* m_ϕ^B *is an* ϵ_T*,*ϵ_R*-approximation of* m_ϕ*.*

Proof. Since the estimates whose probabilities are bounded by Lemma 10 are exactly those estimates used to create m_ϕ^B and $m_{\mathcal{A}}$, when those estimates hold, the bounds between those MDPs hold as well. □

Bounded Sub-optimality

This section presents bounds between the value of a state in $m_{\mathcal{A}}$ and m_ϕ, and then shows that **BFS3** will take only a small number of sub-optimal steps in $m_{\mathcal{A}}$ and, therefore, a small number of sub-optimal steps in m_ϕ.

Lemma 12. *The difference in the values of the two MDPs* $m_{\mathcal{A}}$ *and* m_ϕ *for a given state is bounded by the Simulation Lemma.*

$$|V_{m_{\mathcal{A}}}(s) - V_{m_\phi}(s)| \quad \leq \quad s(\epsilon_T + \epsilon_T^c, \epsilon_R + \epsilon_R^c), \tag{5.16}$$

$$|Q_{m_{\mathcal{A}}}(s,a) - Q_{m_\phi}(s,a)| \quad \leq \quad s(\epsilon_T + \epsilon_T^c, \epsilon_R + \epsilon_R^c), \tag{5.17}$$

$$s(\epsilon_T + \epsilon_T^c, \epsilon_R + \epsilon_R^c) \quad = \quad \frac{(\epsilon_R + \epsilon_R^c) + \gamma V_{\max}(\epsilon_T + \epsilon_T^c)}{1 - \gamma}. \tag{5.18}$$

Proof. Since $m_{\mathcal{A}}$ is an ϵ_T^c, ϵ_R^c approximation of m_ϕ^B, and m_ϕ^B is an ϵ_T, ϵ_R approximation of m_ϕ, $m_{\mathcal{A}}$ is an $(\epsilon_T + \epsilon_T^c)$, $(\epsilon_R + \epsilon_R^c)$ approximation of m_ϕ. Proof follows immediately from the Simulation Lemma. □

Lemma 13. *Acting near-optimally in $m_{\mathcal{A}}$ is sufficient for acting near-optimally in m_{ϕ}. Specifically, an action that is ϵ_A-optimal in $m_{\mathcal{A}}$ is ϵ-optimal in m_{ϕ}, where $\epsilon_A = \epsilon - s(\epsilon_T + \epsilon_T^c, \epsilon_R + \epsilon_R^c)$.*

Proof. Proof is immediate. □

Since an action that is ϵ_A-optimal in $m_{\mathcal{A}}$ is ϵ-optimal in m_{ϕ}, we will bound the number of sub-ϵ_A-optimal steps in $m_{\mathcal{A}}$.

Definition 23. *A* model discovery *is the event where the agent visits an unknown state-action pair.*

Definition 24. *A* value discovery *is the event where **BFS3** plans and causes a belief state to become pinned.*

Definition 25. *A* policy epoch *is a sequence n steps long starting at t, where $\forall_{0<i\leq n}\pi_t = \pi_i$. That is, a policy epoch is a time period during which the policy does not change.*

Definition 26. *A* history epoch *is a sequence of steps without a model discovery.*

Definition 27. *The agent's estimate of the value of belief-state x at time step t is $V_t(x)$. Since **BFS3** will use the estimates from the previous time-step as a way to boot-strap its current estimates, we define $V_{-1}(x) = V_{\max}$.*

Every time **BFS3** visits a belief state for the first time in a given policy epoch, it will run **FSSS** at most CA times, using each of the at most CA possible next belief states as the roots. **FSSS** will reuse and overwrite the previous value estimates, such that if **BFS3** runs **FSSS** on belief-state x at step t, **FSSS** will start with the upper bound on the value of each belief state at $V_{t-1}(x)$, and an equivalent memory for the lower bound. After **FSSS** is done, $V_t(x)$ is the new upper-bound estimate for x.

The following is the proof of Theorem 2

Proof. Let $m^h_{\mathcal{A}}$ be the set of belief states in $m_{\mathcal{A}}$ with the history h. Once an agent leaves $m^h_{\mathcal{A}}$ it can never return, since the only way h can change is by growing bigger.

The strategy will be to show that when **BFS3** enters $m^h_{\mathcal{A}}$, the number of sub-ϵ_A-optimal actions it takes before leaving is small. The time starting when **BFS3** enters this set of states and leaves is a history epoch.

The number of possible model discoveries during a history epoch is 1, since making a discovery ends the epoch.

The number of possible value discoveries during a history epoch is S, since there are at most S possible belief states in the set $m^h_{\mathcal{A}}$, and by the consistency condition, a value discovery can be made once per belief state. In addition, after S value discoveries in a history epoch, all remaining steps in that history epoch will be ϵ_A-optimal, since all states have ϵ_A-accurate values.

Since, at step t, **BFS3** will either replan or the policy will remain unchanged since the last time the agent visited belief-state x_t, either $\operatorname{argmax}_a Q_A(x_t, a)$ is ϵ_A-optimal, or the current policy π_t will reach an unknown state-action pair in the next d steps with probability at least δ_l.

If $\operatorname{argmax}_a Q_A(x_t, a)$ is not ϵ_A-optimal, with probability at least δ_l, the agent will either make a model discovery (if all the d belief states in the sequence remain unknown) or a value discovery (if **BFS3** replans during the sequence and figures out the ϵ_A-accurate value).

The total number of possible model-discovery and value-discovery events during a history epoch, when combined, numbers no more than $S + 1$ (there are S possible value discoveries, and one possible model discovery). Since every d-step window beginning at a belief state with inaccurate Q-values has a probability of at least δ_l of making one of these $S + 1$ discoveries, the expected number of sub-ϵ_A-optimal steps per history epoch is at most $(S+1)d/\delta_l$.

Since there are at most BSA history epochs, one per possible model discovery, the expected number of sub-ϵ_A-optimal steps for the lifetime of the agent is at most $BSA(S+1)d/\delta_l$. $\square$

Chapter 6
Experiments

In this chapter, I present experiments performed by myself and others, examining the **BOSS** and **BFS3** algorithms, and comparing them to other algorithms in the same spirit.

6.1 Parameter Choice

In each experiment, several values for each of the relevant parameters were used, but sometimes only the best were recorded. The exact method of parameter choice is explained for each experiment.

6.1.1 RMAX

Many algorithms are fairly robust with respect to their parameters. For instance, the **RMAX** [3] family of algorithms [4, 29] has one important parameter, m, indicating how many samples of a state-action pair are necessary before that pair is considered *known*. For a given environment, there is usually a value m^* for which any $m \geq m^*$ will cause the **RMAX** agent to find the optimal policy — it needs at least m^* samples of some set of key state-action pairs to know how to behave well. Increasing m beyond m^* will result in a longer period of sup-optimal behavior at the beginning of the trial. Decreasing m will result in potentially defective policies built on too little information.

Since it is easy to identify the optimal value for *m*—increasing *m* does not improve the resulting policy value, and decreasing *m* makes it worse—a brief parameter sweep is sufficient to find the best value for any experiment. Many environments have, for each state, a relatively small number of possible successor states (for instance, in a grid world, those states that are adjacent). And in many of these environments, a single successor state takes the bulk of the probability mass (for instance, if the stochasticity comes from adding a failure probability).

6.1.2 BOSS

BOSS, or **Best Of Sampled Set**, is discussed in detail in Chapter 4.

The **BOSS** algorithm has two important parameters: K indicates how many posterior samples should be used when constructing a hyper-model, and B indicates how many samples of a particular state-action pair are required before that pair becomes *known*, triggering the creation of a new hyper-model.

Unlike **RMAX**, where a too-small value for m can effect reliably poor policies, the B parameter for **BOSS** is much more robust. When a state-action pair becomes *known* according to **RMAX**, the agent decides that there is nothing more to be learned for that pair and may avoid it entirely in the future. Without enough samples, its initial estimation of that state-action's dynamics may be flawed. With **BOSS**, the agent is not drawn specifically to *unknown* state-action pairss, but instead is drawn to state-action pairs whose dynamics have high variance in the posterior. So, even if B is set to something too small, **BOSS** can still be drawn to the *unknown* parts of the MDP because posterior variance and B are uncorrelated. This is not to say that **BOSS** will work equally well with any value for B — if B is too small, the posterior might still have too much variance when the last hyper-model is constructed, causing it to have exploration built in to its final policy (leading to sub-optimal steps for eternity). If B is too large, the effect is similar to that seen with **RMAX**: the exploration phase at the beginning of the experiment may last longer than it needs to, leading to a lower cumulative reward.

The K parameter for **BOSS**, which chooses how many posterior samples to use when creating the hyper-model, must fall within a certain range to be useful. In the limit, as $K \to \infty$, **BOSS** will break down completely. Since **BOSS** views all of the posterior samples going into a particular hyper-model equally, sampling an infinite amount of them will lead to a hyper-model that is completely uninformed by the agent's experience. Conditioning on observations causes the posterior to change shape, making some models more likely than others. If too many samples are taken, then the models that would normally be skipped, due to low probability, will still be represented in the hyper-model. In practice, the value of K that becomes "too big" is terrifically large, since with enough data the posterior can become extremely peaked.

Setting K too low can cause underexploration. If there are not enough posterior samples used to create the hyper-model, it is possible that some state-action pairs will have a *pessimistic* value in the hyper-model, causing the agent to sometimes actively avoid high-value states when they are *unknown*, since they will have high variance in the posterior. **Bayesian DP** [20], discussed in Section 3.3, is similar to **BOSS** with $K = 1$, except that it addresses the under-exploration issue with periodic resampling.

Finding ideal parameters for **BOSS** is, in general, easy to do. Although the theoretical guarantees require K to be large enough such that all unknown state-action pairs have optimistic values, in practical settings it's only necessary that most of them do, so that **BOSS** has exploration incentives.

6.1.3 BFS3

BFS3, or **Bayesian Forward Search Sparse Sampling**, is discussed in detail in Chapter 5. It leans heavily on **FSSS**, which is discussed in detail in Chapter 3, Section 3.13.

The choice of parameters for **BFS3** is, for the most part, pre-specified by **FSSS**. Since, for **BFS3** to be effective, **FSSS** must be able to find the value of any state in the "true" model, **BFS3** must choose parameters that are at least as conservative as those needed by **FSSS**.

The only parameter that will often need to be *more* conservative than the one required by **FSSS** is the C: how many times **FSSS** will try each action from each visited state when doing simulations. There are MDPs for which **FSSS** only needs $C = 1$ to try out each action for each visited state — if a model is deterministic, then the action will give the same results each time. However, a deterministic "true" MDP does not induce a deterministic BAMDP: There can still be stochasticity in the posterior until the agent has observed a transition from the state being visited by **FSSS**. As a result, C must often be higher than the minimum needed for **FSSS** to be effective for the "true" model.

Increasing parameters to values above what are needed will not harm the performance of the agent with respect to sample complexity, but can have an effect on the computation required. Computation will grow linearly with the number of trajectories run, the number of times that **FSSS** will try each action for each visited state, and the search depth.

6.2 BOSS Experiments

This section lists experiments in which the **BOSS** algorithm was compared to other model-based algorithms.

6.2.1 Chain

Consider the well-studied 5-state chain problem (*Chain*) [17, 20]. The agent has two actions: Action 1 advances the agent along the chain, and Action 2 resets the agent to the first node. Action 1, when taken from the last node, leaves the agent where it is and gives a reward of 10—all other rewards are 0. Action 2 always has a reward of 2. With probability 0.2 the outcomes of the two actions are switched, however. Optimal behavior is to always choose Action 1 to reach the high reward at the end of the chain.

The slip probability 0.2 is the same for all state–action pairs. Poupart et al. (2006) consider the impact of encoding this constraint into the prior on the transition dynamics. That is, whereas in the Full prior the agent assumes each state–action pair corresponds to independent multinomial distributions over next states, under the Tied prior, the agent knows the underlying transition dynamics except for the value of a single slip probability that is shared between all state–action pairs. They also introduce a Semi prior in which the two actions have independent slip probabilities. Posteriors for Full can be maintained using a Dirichlet (the conjugate for the multinomial) and Tied/Semi can be represented with simple Beta distributions.

In keeping with published results on this problem, Table 6.1 reports cumulative rewards in the first 1000 steps, averaged over 500 runs. Standard error is on the order of 20 to 50. The optimal policy for this problem scores 3677. The

Table 6.1: Cumulative reward in *Chain* [1]

	Tied	Semi	Full
BEETLE	3650	3648	1754
exploit	3642	3257	3078
BOSS	3657	3651	3003
RAM-RMAX	3404	3383	2810
Bayesian-DP			3158

exploit algorithm is one that always acts optimally with respect to the average posterior-weighted model. **RAM-RMAX** [5] is a version of **RMAX** that can exploit the tied parameters of tasks like this one. Results for **BEETLE** and **exploit** are due to Poupart et al. (2006). All runs used a discount factor of $\gamma = 0.95$ and **BOSS** used $B = 10$ and $K = 5$.

The parameter values for **BOSS** were an initial guess, and the guess allowed **BOSS**'s performance to either match or exceed every other algorithm in the experiment.

All algorithms perform very well in the Tied scenario (although **RAM-RMAX** is a bit slower as it needs to estimate the slip probability very accurately to avoid finding a suboptimal policy). Poupart et al. (2006) point out that **BEETLE** is more effective than **exploit** (an undirected approach) in the Semi scenario, which requires more careful exploration to perform well. In Full, however, **BEETLE** falls behind because the value function is difficult to approximate in high dimensional spaces.

BOSS, on the other hand, explores as effectively as **BEETLE** in Semi, but is also effective in Full. A similarly positive result in Full is obtained by **Bayesian DP** [20].

Bayesian Modeling of State Clusters

The idea of state clusters is implicit in the Tied prior. We say that two states are in the same cluster if their probability distributions over relative outcomes are the same given any action. In *Chain*, for example, the outcomes are advancing along the chain or resetting to the beginning. Both actions produce the same distribution on these two outcomes independent of state, Action 1 is 0.8/0.2 and Action 2 is 0.2/0.8, so *Chain* can be viewed as a one-cluster environment.

We introduce a variant of the chain example, the two-cluster *Chain2*, which includes an additional state cluster. Cluster 1—states 1, 3, and 5—behaves identically to the cluster in Chain. Cluster 2—states 2 and 4—has roughly the reverse distributions (Action 1 0.3/0.7, Action 2 0.7/0.3).

RAM-RMAX can take advantage of cluster structure, but only if it is known in advance. In this section, we show how **BOSS** with an appropriate prior can learn an unknown cluster structure and exploit it to speed up learning. The prior used is the cluster prior, defined in Chapter 2.

We ran **BOSS** in a factorial design where we varied the environment (*Chain* vs. *Chain2*) and the prior (Tied, Full, vs. Cluster). For our experiments, **BOSS** used a discount factor of $\gamma = 0.95$, knownness parameter $B = 10$, and a sample size of $K = 5$. The Cluster CRP used $\alpha = 0.5$ and whenever a sample was required the Gibbs sampler ran for a burn period of 500 sweeps with 50 sweeps between each sample.

Figure 6.1 displays the results of running **BOSS** with different priors in *Chain* and *Chain2*. The top line on the graph corresponds to the results for *Chain*. Moving from left to right, **BOSS** is run with weaker priors—Tied, Cluster, and Full. Not surprisingly, performance decreases with weaker priors. Interestingly, however, Cluster is not significantly worse than Tied—it is able to

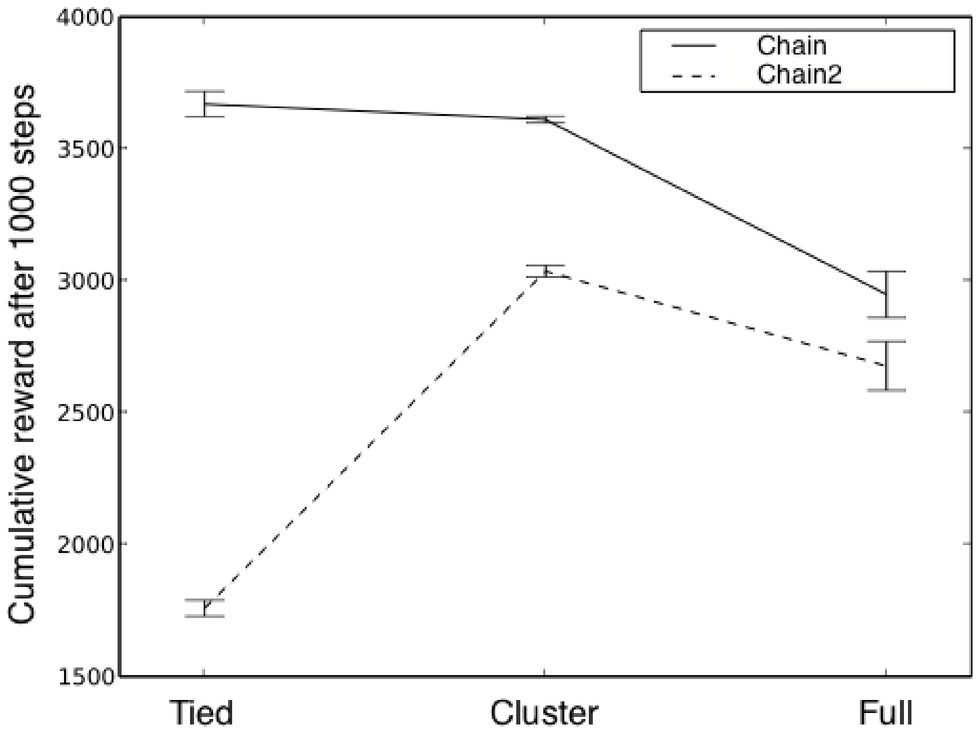

Figure 6.1: Varying priors and environments in **BOSS**.

identify the single cluster and learn it quickly.

The second line on the plot is the results for *Chain2*, which has two clusters. Here, Tied's assumption of the existence of a single cluster is violated and performance suffers as a result. Cluster outperforms Full by a smaller margin, here. Learning two independent clusters is still better than learning all states separately, but the gap is narrowing. On a larger example with more sharing, we'd expect the difference to be more dramatic. Nonetheless, the differences here are statistically significant (2×3 ANOVA $p < 0.001$).

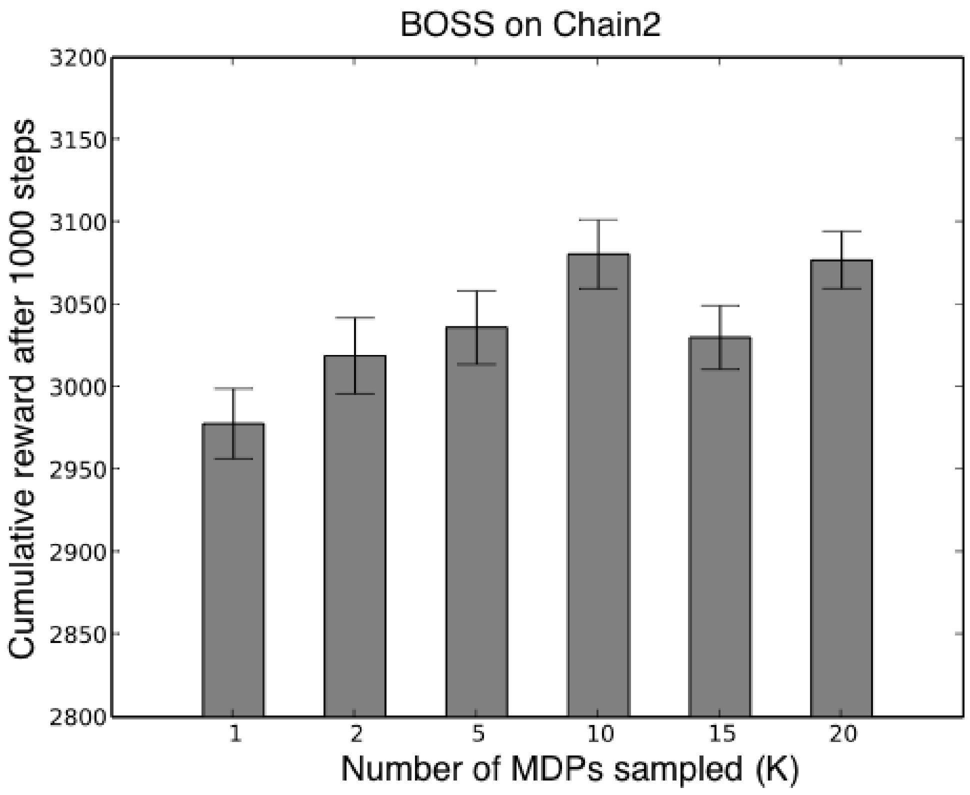

Figure 6.2: Varying K in **BOSS**.

Varying K

The experiments reported in the previous section used model samples of size $K = 5$. Our next experiment was intended to show the effect of varying the sample size. Note that **Bayesian DP** is very similar to **BOSS** with $K = 1$, so it is important to quantify the impact of this parameter to understand the relationship between these algorithms.

Figure 6.2 shows the result of running **BOSS** on *Chain2* using the same parameters as in the previous section. Note that performance generally improves with K. The difference between $K = 1$ and $K = 10$ is statistically significant (t-test $p < 0.001$).

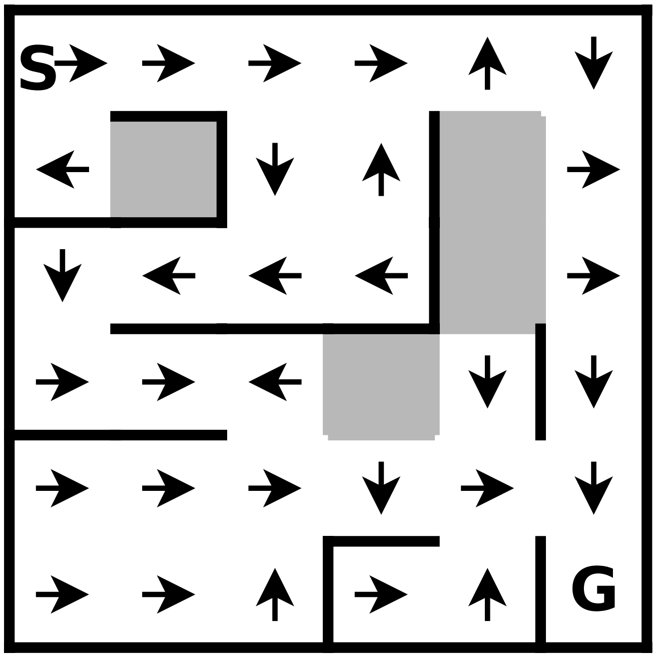

Figure 6.3: Diagram of *6x6 Marble Maze*.

6.2.2 Marble Maze

To demonstrate the exploration behavior of our algorithm, we developed a 6x6 grid-world domain with standard dynamics [41]. In this environment, the four actions, *N*, *S*, *E* and *W*, carry the agent through the maze on its way to the goal. Each action has its intended effect with probability .8, and the rest of the time the agent travels in one of the two perpendicular directions with equal likelihood. If there is a wall in the direction the agent tried to go, it will remain where it is. Each step has a cost of 0.001, and terminal rewards of -1 and $+1$ are received for falling into a pit or reaching the goal, respectively. The map of the domain, along with its optimal policy, is illustrated in Figure 6.3.

The dynamics of this environment are such that each local pattern of walls (at most 16) can be modeled as a separate cluster. In fact, fewer than 16 clusters appear in the grid and fewer still are likely to be encountered along an optimal trajectory. Nonetheless, we expected **BOSS** to find and use a larger set of clusters than in the previous experiments.

For this domain, **BOSS** used a discount factor of $\gamma = 0.95$ and a CRP hyperparameter of $\alpha = 10$. Whenever an MDP set was needed, the Gibbs sampler ran for a burn period of 100 sweeps with 50 sweeps between each sample. We also ran **RMAX** in this domain.

The cumulative reward achieved by the **BOSS** variants that learned the cluster structure, in Figure 6.4, dominated those of **RMAX**, which did not know the cluster structure. The primary difference visible in the graph is the time needed to obtain the optimal policy. Remarkably, **BOSS** $B = 10$ $K = 10$ latches onto near optimal behavior nearly instantaneously whereas the **RMAX** variants required 50 to 250 trials before behaving as well. This finding can be partially explained by the choice of the clustering prior and the outcomes it drew from, which effectively put a lower bound on the number of steps to the goal from any state. This information made it easy for the agent to ignore longer paths when it had already found something that worked.

Looking at the clustering performed by the algorithm, a number of interesting features emerge. Although it does not find a one-to-one mapping from states to patterns of walls, it gets very close. In particular, among the states that are visited often in the optimal policy and for the actions chosen in these states, the algorithm groups them perfectly. The first, third, fourth, and fifth states in the top row of the grid are all assigned to the same cluster. These are the states in which there is a wall above and none below or right, impacting the success probability of N and E, the two actions chosen in these states. The first, second, third, and fifth states in the rightmost column are similarly grouped together.

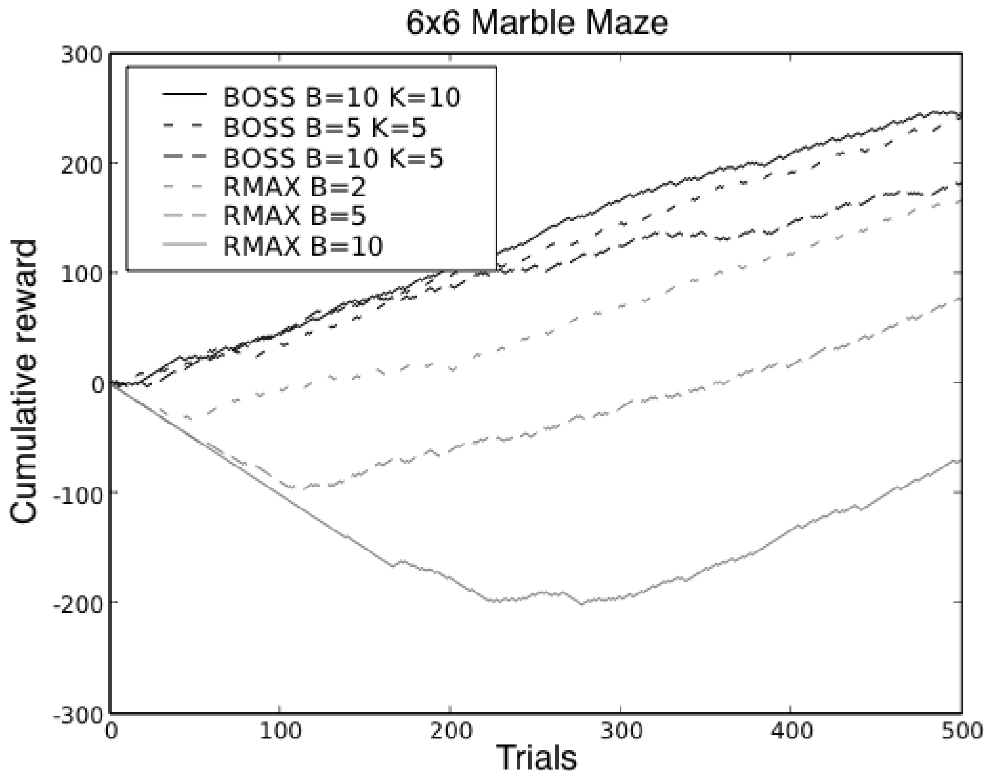

Figure 6.4: Comparison of algorithms on *6x6 Marble Maze*.

These are the states with a wall to the right, but none below or left, impacting the success probability of *S* and *E*, the two actions chosen in these states. Other, less commonly visited states, are clustered somewhat more haphazardly, as it was not necessary to visit them often to obtain high reward in this grid. The sampled models used around 10 clusters to capture the dynamics.

A variety of parameters were used for **BOSS**, all resulting similarly valued policies and unnoticable purely exploration periods, suggesting that **BOSS**'s performance is quite robust with respect to its parameters.

6.2.3 Puddle World

Puddle World [42] is an RL benchmark in which an agent attempts to navigate two-dimensional space using four actions equivalent to *north*, *east*, *south* and *west*. The outcome dynamics are consistent across the entire domain, but the reward associated with each step can vary; in the center of the environment is a large “puddle”, and if the agent ventures into the puddle it receives very poor reward. It is an action-penalty domain where the agent’s goal is to reach a terminal state in one corner. This domain has outcomes with small additive noise.

ROAR/**BOSS** is compared against **Fitted-RMAX** in Figure 6.5. Here, both algorithms are able to find effective policies, but the sub-optimal steps taken by **Fitted-RMAX** significantly hurt its total reward gathered; this poor behavior at the beginning is typical of **RMAX**-based algorithms. ROAR/**BOSS**, on the other hand, is able to start being effective more quickly, since the dynamics of of the model are partly encoded in the prior, though the details are left out. Occasionaly, ROAR/**BOSS** decides to learn about a new area of the state-space for a short period. Such episodes are visible in the plot as sharp, but short, dips downwards as it takes sub-optimal actions.

Although only a single parameter choice is shown in the experiment, it was able to significantly outperform its competitor, and has a very short learning period at the beginning, with other smaller dips later on as it discovers other interesting areas of the environment.

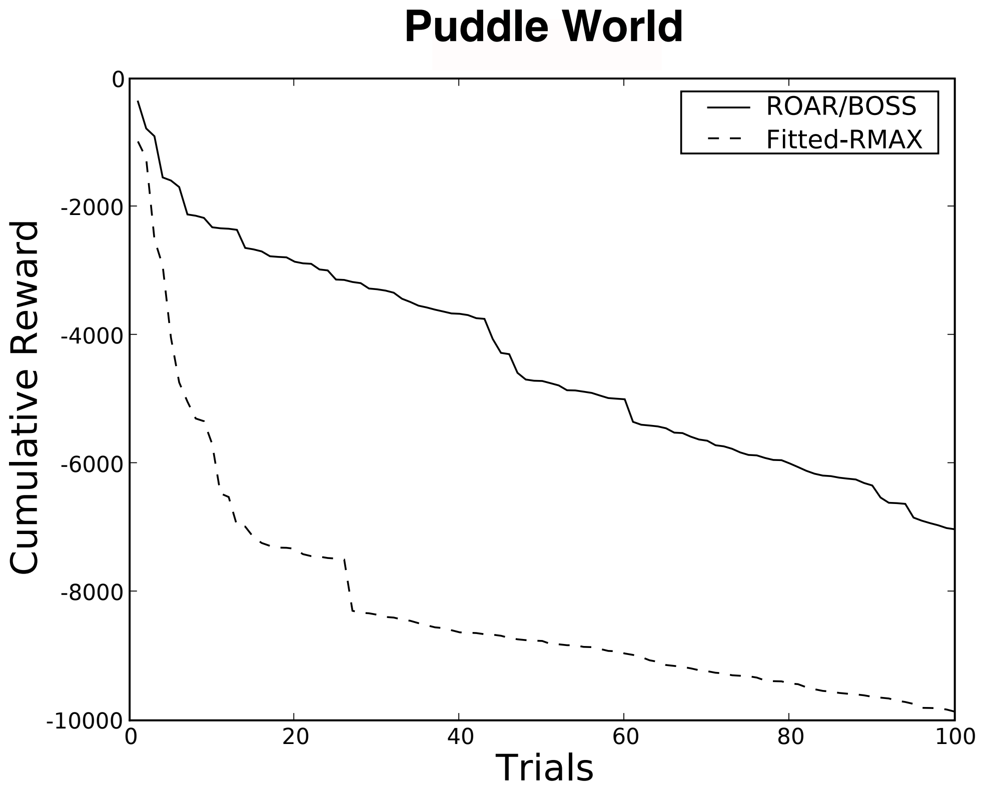

Figure 6.5: ROAR/**BOSS** and **Fitted-RMAX** are compared on the *Puddle World* domain. **Fitted-RMAX** was run with width parameter $b = 0.05$ and knownness threshold $B = 2$. ROAR/**BOSS** was run with $K = 1, B = 5, \alpha = 1.0, \Psi_S = 0.1I, m_S = 200, \Psi_O = 0.01I, m_O = 30, \Psi_r = 1.0I, m_r = 20, \alpha_\phi = 0.001, \beta_\phi = 0.01$. Averaged over 20 runs.

6.2.4 Bridge of Fire

The previous section described a model that could be characterized as "deterministic with small noise". In the *Bridge of Fire* domain, this assumption is violated by making the noise associated with each action determine the action's effectiveness.

In the *Bridge of Fire* domain, the agent must navigate across a bridge, without falling off the side (and into the fire), using three actions corresponding to *walk*, *hand-spring* and *pogo-stick*. *Walk* is a very consistent action, moving the agent forward along the bridge with no lateral noise. *Hand-spring* will move the agent faster, but there is a chance that the agent moves to one side or the other. *Pogo-stick* will take the agent a long way, but there is a huge amount of uncertainty in the movement and more often than not the agent will pogo right off of the bridge.

More concretely, the bridge is a 2×10 rectangle, and the agent begins in the middle of one edge. The *walk* action moves the agent forward according to $N(0.5, 1)$ and sideways according to $N(0, 0)$. The *hand-spring* action moves forward with $N(1, 1)$ and sideways with $N(0, 0.5)$, and finally the *pogo-stick* action moves forward with $N(1.5, 1)$ and sideways with $N(0, 1)$. If the agent makes it to the other side of the bridge, it receives a reward of 100 and the trial ends. If the agent falls, it receives a reward of -100 and the trial ends. Every non-terminal step gives the agent a reward of -1.

Ignoring the noise in the *Bridge of Fire* will cause the agent to prefer *pogo-stick*, since the outcome moves the agent the furthest along the length of the bridge and has zero mean lateral displacement. In reality, the agent should prefer *walk* until near the end, when *hand-spring* becomes safer.

ROAR/**BOSS** is compared against **Fitted-RMAX** and **CORL** in Figure 6.6. We see that the algorithms ROAR/**BOSS** and **CORL** were able to learn the *Bridge of Fire* domain effectively. **Fitted-RMAX** makes assumptions that do not hold in this environment, namely that there is no noise in the outcome distribution, and does poorly as a result because the action it likes the best has enough noise to knock the agent off the bridge.

For **BOSS**'s parameter choices, even the trivial value of $K = 1$ was enough for the agent to be able to find the optimal policy more quickly than **CORL**,

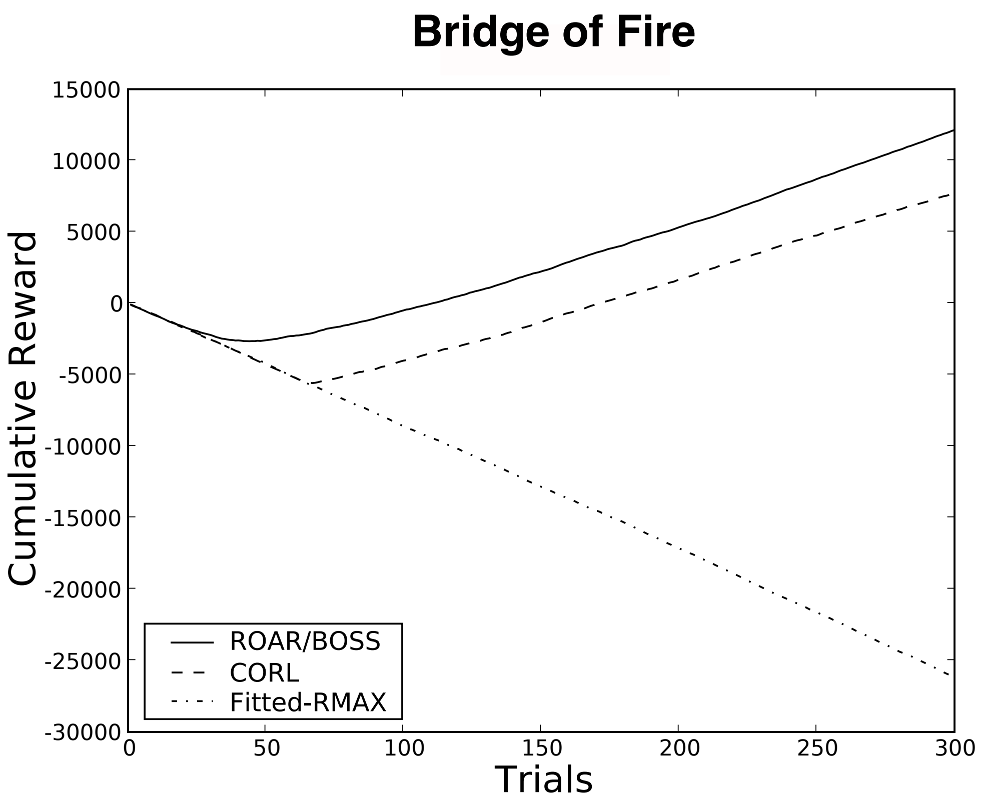

Figure 6.6: ROAR/**BOSS** is compared against **CORL** on the *Bridge of Fire* domain. **CORL** was run with knownness threshold $B = 20$, and was given the correct type function. **Fitted-RMAX** was run with width parameter $b = 0.5$ and knownness threshold $B = 3$. ROAR/**BOSS** was run with $K = 1, B = 10, \alpha = 1.0, \Psi_{\text{S}} = 0.1I, m_{\text{S}} = 3, \Psi_{\text{O}} = 0.1I, m_{\text{O}} = 3, \Psi_{\text{r}} = 1.0I, m_{\text{r}} = 50, \alpha_{\phi} = 0.001, \beta_{\phi} = 0.005$. Averaged over 20 runs.

which is a more conservative algorithm. The rest of the parameters are associated with ROAR, rather than **BOSS**, and are chosen such that the prior guesses about noise are within an order of magnitude of the truth.

6.2.5 Mind the Gap

The *Mind the Gap* domain is similar to *Bridge of Fire*, except none of the actions have lateral displacement; the agent always goes straight and will never fall off the side. However, beginning 5 units from the initial state, there is a 1 unit stretch where the bridge has given out. To compensate, the agent has that the algorithm controls has become better at the using the *pogo-stick* action: The agent will either remain still or move forward exactly 1.5 units. The *pogo-stick* action becomes appropriate when the agent is within 0.5 units of the gap, though due to noise in the other actions, it is still very difficult to avoid the gap with high probability.

The correct model in *Mind the Gap* is very difficult to learn. Using ROAR with **Bayesian DP** proved ineffective; sampling only a single model at a time didn't give the agent enough reason to learn sufficiently about some parts of the state space. However, using **BOSS** as the sampler was more successful. **BOSS** is a very conservative algorithm, "pooling" the optimism from each of its sampled models for planning. This domain demonstrates the need for such optimism.

ROAR/**BOSS** and ROAR/**Bayesian DP** are compared against **CORL** in Figure 6.7. Because of its unimodal outcome distribution assumption, **CORL** was unable to perform well in the *Mind the Gap* domain, even after knowing the correct "type" function for clustering. ROAR, with no prior knowledge of the clustering, is able to effectively learn multi-modal distributions and can correctly model this environment's dynamics.

For this experiment, an effect was observed by changing **BOSS**'s K parameter. Since the environment is fairly difficult, increasing K—which can be likened to increasing how conservative **BOSS** is—caused **BOSS** to remain optimistic longer for some of the harder-to-learn actions. **Bayesian DP**, which is very similar to **BOSS** with $K = 1$, was unable to find the optimism required to try the *pogo-stick* action when it was needed.

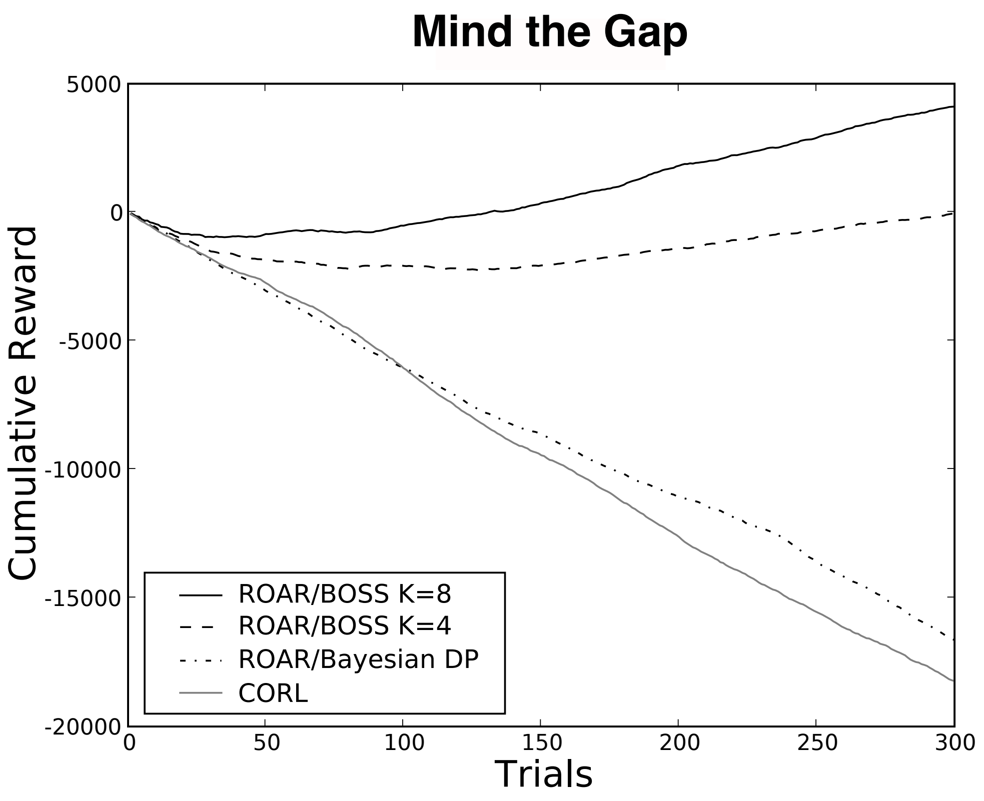

Figure 6.7: ROAR/**BOSS**, ROAR/**Bayesian DP**, and **CORL** are compared on the *Mind the Gap* domain. **CORL** was run with knownness threshold $B = 10$. ROAR algorithms were run with $B = 1, \alpha = 1.0, \Psi_S = 0.5I, m_S = 10, \Psi_O = 0.1I, m_O = 3, \Psi_r = 0.01I, m_r = 50, \alpha_\phi = 0.01, \beta_\phi = 0.001$. ROAR/**Bayesian DP** drew samples every 20 steps. Averaged over 20 runs.

6.3 BFS3 Experiments

To demonstrate **BFS3**, we will show its performance in a number of domains, and show how the use of different priors can affect its performance. This flexibility with respect to using different priors is a compelling reason to use MCTS algorithms in general, and **BFS3** in particular, for model-based reinforcement learning.

The first experiment is very simple, and exact Bayes-optimal behavior is possible. *Bernoulli-bandits* is a 5-armed bandit problem where each arm a has some unknown probability $p_a \sim Beta(\alpha, \beta)$ of returning a reward of 1 or 0 each step. For each run in the experiment, the environment parameters p_a are drawn directly from the prior. The Gittins Index [43] can be used to achieve Bayes-optimal behavior for multi-armed bandit problems. After 1000 steps, **BFS3** with the true prior received an average (over 40 runs) total reward of 727, where the Bayes-optimal agent received an average total reward of 815.

When using the FDM prior with a small state space, **BEB** may be considered a better choice. Since **BEB** operates greedily according to a grounded MDP, rather than a belief-MDP, planning is made potentially much easier. This algorithm is limited, however, in that it requires a known reward function; it can only deal with uncertainty in the transition function.

6.3.1 Reward Priors

BFS3, with the right prior, can handle unknown rewards. In many domains, there are only a few possible reward values. For example, many path-finding domains give a reward of -1 for all actions. Or, there is a reward for a particular outcome that can be achieved from multiple states: these states would share the same reward value. To represent this common structure in a generative model, the Dirichlet Process (DP) [44] may be used:

$$R_{s,a} \sim \mathrm{DP}(\alpha, \mathrm{Unif}(R_{\min}, R_{\max})).$$

Note that with this prior, rewards are deterministic, if unknown.

In Figure 6.8, **BFS3**, with the unknown reward prior, is shown to suffer no significant performance penalty compared to **BFS3** with the known-reward prior and to **BEB** (which also uses the known-reward prior). The domain is

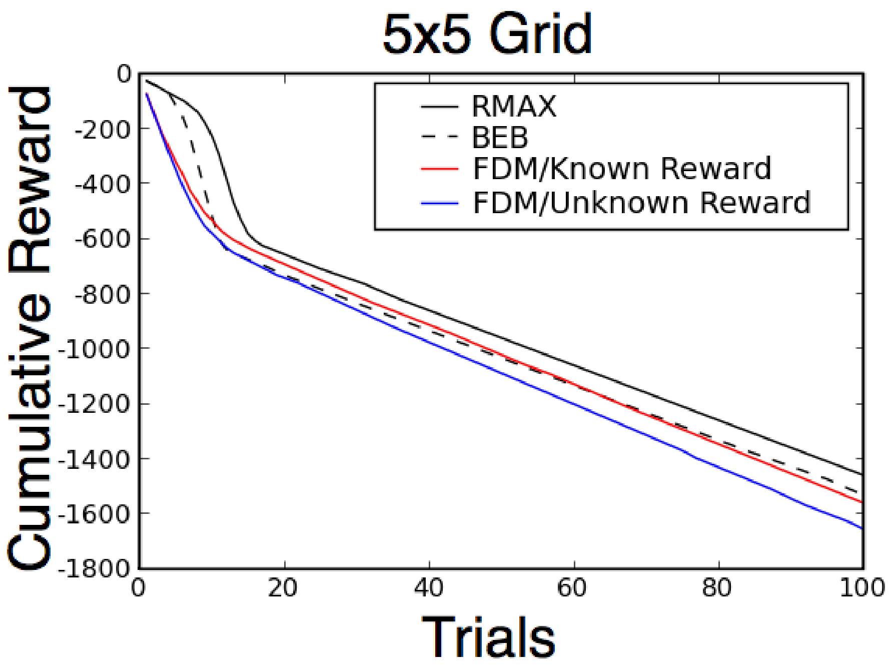

Figure 6.8: Here, **BFS3**/FDM, with both known rewards and unknown rewards with a DP prior, is compared against the known-reward **BEB**, and **RMAX** [3], a well known algorithm with PAC-MDP guarantees, with $M = 5$. Results are averaged over 40 runs.

a 5×5 grid world, where the agent must find its way from one corner to the other. The agent can choose to go north, east, south or west, and with probability 0.2 it will go in a direction perpendicular to the one intended.

6.3.2 Factored Priors

Along with FDM, we introduce the Factored-Object prior, which describes factored MDPs in which the state features are broken up into a number of independent yet identical objects. The action also has two features: the first indicates which object is being acted upon, and the second indicates which action is being performed. Factored-Object essentially has a single FDM posterior, which it applies to each object in the state simultaneously, sharing both information (for faster convergence) and memory.

For a single object, Factored-Object and FDM are the same. For two objects, FDM has to learn separately how a particular action affects a particular object for every possible configuration of objects—for a different state of an object not being operated on, FDM must re-learn how the original object is affected. Factored-Object allows the agent to learn about multiple objects at the same time: it knows that a given action affects object 1 in the same way it affects object 2, and generalizes appropriately.

The Paint/Polish world [45] provides a situation where the simple and convenient FDM prior is insufficient. The size of the state-space grows exponentially with the number of cans to paint (each of which introduces four binary features). Figure 6.10 shows the results with a single can (and 2^4 states) and the results with two cans (and $2^{4 \cdot 2}$ states). Figure 6.11 shows the results with four cans (and 2^{4^4} states). In these experiments, we see how encoding knowledge into the algorithm's prior can dramatically affect its performance. The FDM prior simply does not scale with the number of parameters, and the Factored-Object prior allows the agent to correctly generalize its experience without trying to visit each state.

The parameters for **BFS3**, which are simply the parameters provided to **FSSS**, were chosen by searching for parameters that allowed **FSSS** to find accurate values on the known MDP, which is relatively small and easy to solve with value iteration.

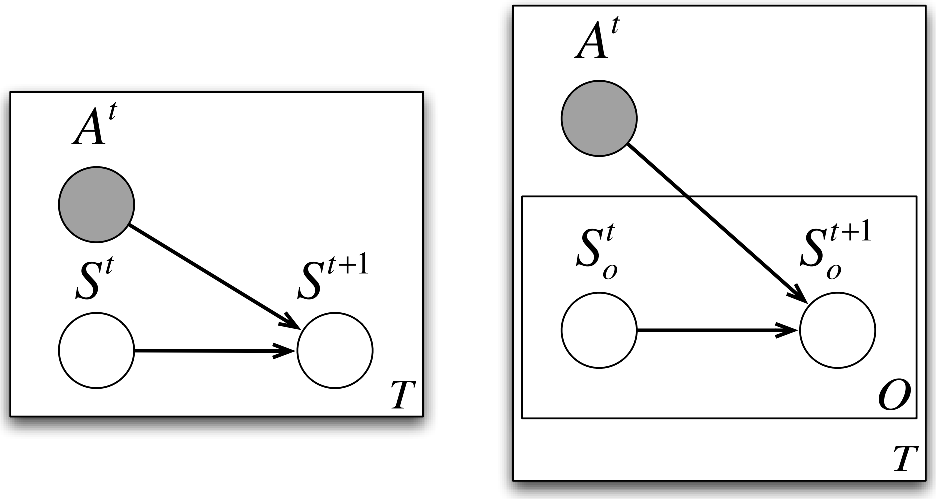

Figure 6.9: The Directed Graphical Models for FDM (left) and Factored-Object (right). Nodes indicate random variables, and edges show possible dependencies. Shaded nodes are observed. Boxes indicate repetition.

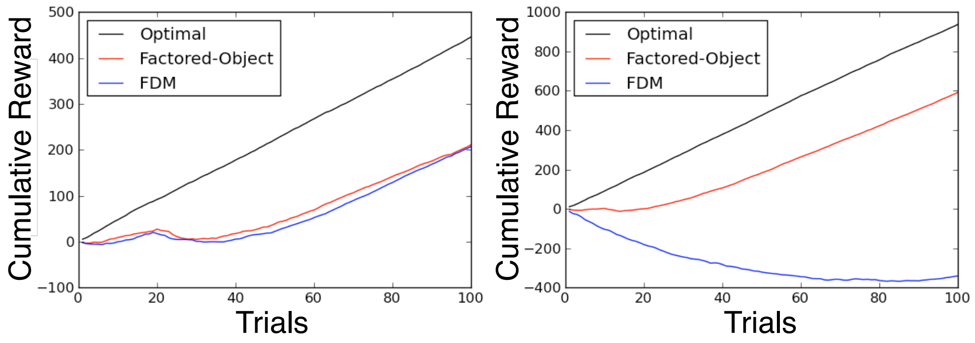

Figure 6.10: **BFS3** with FDM and Factored-Object priors. **Left:** Paint/Polish with 1 can. FDM and Factored-Object are identical for one object, and give the same performance. **Right:** Paint/Polish with 2 cans. Factored-Object outperforms FDM because it is better able to generalize. Results are averaged over 40 runs.

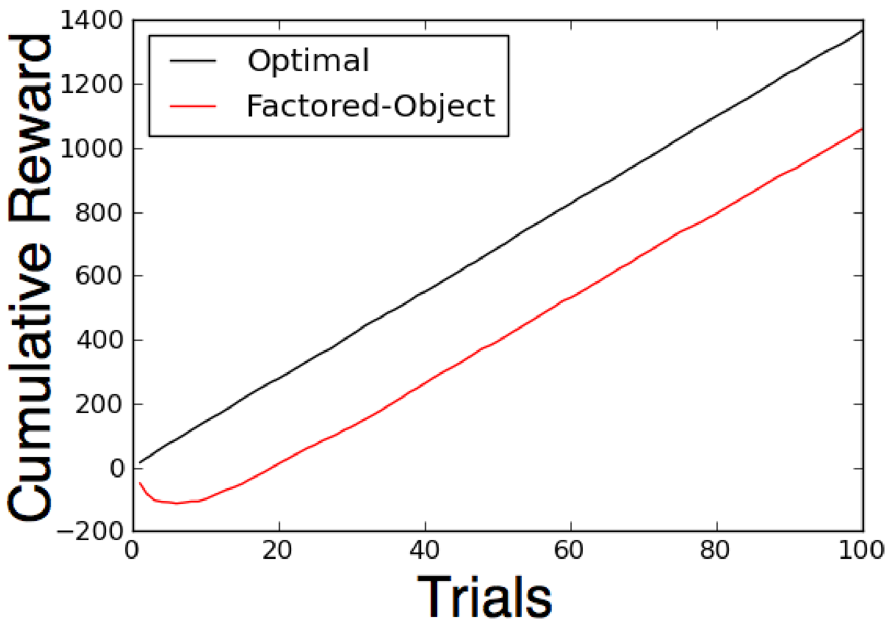

Figure 6.11: Paint/Polish with 4 cans. Using the Factored-Object prior allows **BFS3** to learn quickly despite the very large state space. Using FDM does not allow learning in a reasonable amount of time, and is not pictured. Results are averaged over 40 runs.

6.3.3 Wumpus World

BFS3 can also be used to apply Bayesian modeling to POMDPs. *Wumpus World* [41] is based on a classic computer game in which an agent wanders through a 4×4 maze filled with fog, making it impossible to see past its current cell. Even though the agent cannot see, it can feel a breeze if there is a pit in an adjacent cell, and smell a stench if there is a Wumpus[1] nearby. If the agent falls into a pit, it will remain there forever. If the agent runs into the Wumpus, it is eaten forever. If the agent shoots its one arrow in the direction of the Wumpus, the Wumpus is slain. If the arrow misses the Wumpus, the trial ends and presumably the agent goes home. We replicate the dynamics presented in detail by Sorg et al. (2010).

Wumpus World is based on a deterministic process, but since the agent only knows attributes of cells that it has visited, it appears stochastic. The prior over different possible mazes is known and given to the agent, and from this prior it can infer the correct posterior distribution over what happens when it performs a particular action in a particular belief-state.

We ran **BFS3** on *Wumpus World* with a search depth of 15, and varied the number of trajectories per step. Agents with 500, 1000, and 5000 trajectories per step averaged 0.267, 0.358 and 0.499 cumulative reward, respectively. Averages were taken over 1000 attempts. We compare these values to a variance-based reward bonus strategy, **Variance-based BEB** [21], which, when tuned, averaged 0.508. The **Variance-based BEB** results are due to Sorg et al. 2010.

That **BFS3** performs better in *Wumpus World* as the computation budget

[1] A Wumpus is a monster that eats RL agents.

is increased supports our argument that the algorithm has a *computational resources* knob which, when tuned higher, causes the agent's behavior to get closer to being Bayes-optimal at the cost of decision-making speed.

6.3.4 Texture World

Texture World is a simple grid world with four actions: *north*, *east*, *south*, and *west*. From any state, the action attempts to move the agent one cell in the desired direction, as long as the destination is in the grid and is not a *black* cell.

If the destination is valid, the action succeeds with some probability that is a function of the agent's current cell. Cells can be *red*, *green*, *blue*, or *black*. If the world is textured, then all cells of the same color have the same success probability, drawn from the Beta distribution. If the world is not textured, then each state has its own success probability, independently sampled from the Beta distribution. The world is either textured or not textured with some probability.

The exact model is described by a generative process,

$$\begin{aligned} T &\sim \mathrm{Flip}(p), && (6.1) \\ \rho_s &= \begin{cases} \theta^C_{C_s} & : \; T = \text{heads} \\ \theta^S_s & : \; T = \text{tails}, \end{cases} && (6.2) \\ \theta^C_c &\sim \mathrm{Beta}(\alpha, \beta), && (6.3) \\ \theta^S_s &\sim \mathrm{Beta}(\alpha, \beta), && (6.4) \end{aligned}$$

where ρ_s is the success probability in state s, C_s is the color of state s, p is the prior likelihood that the world is textured, and α and β are hyperparameters for the Beta prior. The colors themselves are not part of the prior—the agent knows the color of every state.

So sample a model from this posterior, first the latent parameter T must be sampled. Since there are only two possible values for T, *heads* or *tails*, it is sufficient to find their likelihood ratios to pick from the posterior.

With observations o,

$$P(T|o) \quad \propto \quad P(o|T)P(T). \tag{6.5}$$

Evaluating $P(o|T = \text{tails})$ is straightforward: each of the states are a separate learning problem equivalent to learning the bias of a coin. Let n_s be the number of successes from state s, and let t_s be the total number of attempts from state s. Then,

$$P(o|T = \text{tails}) \quad = \quad \prod_s \int_\theta \text{Beta}(\theta|\alpha,\beta)\text{Bin}(n_s|\theta,t_s)d\theta, \tag{6.6}$$

and Equation 2.20 shows how this turns into

$$P(o|T = \text{tails}) \quad = \quad \prod_s \frac{(t_s)!}{n_s!(t_s-n_s)!}\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\frac{\Gamma(\alpha+n_s)\Gamma(\beta+t_s-n_s)}{\Gamma(\alpha+\beta+t_s)}. \tag{6.7}$$

Evaluating $P(o|T = \text{heads})$ is a little trickier: groups of states have their parameters tied together, but the problem cannot be considered one Multinomial distribution for each cluster. Instead, it is still one multinomial for each state, integrated across the θ for each cluster.

$$P(o|T = \text{heads})$$

$$= \prod_c \int_\theta \text{Beta}(\theta|\alpha,\beta) \prod_{s \in c} \text{Bin}(n_s|\theta,t_s)d\theta, \tag{6.8}$$

$$= \prod_c \int_\theta \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{(\alpha-1)}(1-\theta)^{(\beta-1)} \prod_{s\in c}\frac{(t_s)!}{n_s!(t_s-n_s)!}\theta^{n_s}(1-\theta)^{(t_s-n_s)}d\theta, \tag{6.9}$$

$$= \prod_c \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\prod_{s\in c}\frac{(t_s)!}{n_s!(t_s-n_s)!}\int_\theta \theta^{(\alpha-1)}(1-\theta)^{(\beta-1)}\prod_{s\in c}\theta^{n_s}(1-\theta)^{(t_s-n_s)}d\theta, \tag{6.10}$$

$$= \prod_c \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\prod_{s\in c}\frac{(t_s)!}{n_s!(t_s-n_s)!}\int_\theta \theta^{(\alpha+\sum_{s\in c} n_s-1)}(1-\theta)^{(\beta+\sum_{s\in c}(t_s-n_s)-1)}d\theta. \tag{6.11}$$

Let $n_c = \sum_{s \in c} n_s$, and $t_c = \sum_{s \in c} t_s$. Then,

$$\begin{aligned}
&P(o|T = \text{heads}) \\
&= \prod_c \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \prod_{s\in c} \frac{(t_s)!}{n_s!(t_s-n_s)!} \int_\theta \theta^{(\alpha+n_c-1)}(1-\theta)^{(\beta+(t_c-n_c)-1)} d\theta, &(6.12)\\
&= \begin{array}{l} \prod_c \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \prod_{s\in c} \frac{(t_s)!}{n_s!(t_s-n_s)!} \\ \cdot \int_\theta \frac{\Gamma(\alpha+n_c)\Gamma(\beta+t_c-n_c)}{\Gamma(\alpha+\beta+t_c)} \frac{\Gamma(\alpha+\beta+t_c)}{\Gamma(\alpha+n_c)\Gamma(\beta+t_c-n_c)} \theta^{(\alpha+n_c-1)}(1-\theta)^{(\beta+(t_c-n_c)-1)} d\theta, \end{array} &(6.13)\\
&= \begin{array}{l} \prod_c \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \prod_{s\in c} \frac{(t_s)!}{n_s!(t_s-n_s)!} \\ \cdot \frac{\Gamma(\alpha+n_c)\Gamma(\beta+t_c-n_c)}{\Gamma(\alpha+\beta+t_c)} \int_\theta \text{Beta}(\theta|\alpha+n_c, \beta+t_c-n_c) d\theta, \end{array} &(6.14)\\
&= \prod_c \frac{\Gamma(\alpha+n_c)\Gamma(\beta+t_c-n_c)}{\Gamma(\alpha+\beta+t_c)} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \prod_{s\in c} \frac{(t_s)!}{n_s!(t_s-n_s)!}. &(6.15)
\end{aligned}$$

Although tricky to derive, these posterior likelihoods are easy to compute. Once T is sampled (that is, a choice is made about whether or not the world is textured), the success likelihoods of the clusters or of the states, whichever is appropriate, is taken from the posterior Beta distribution that accounts for the successes and failures seen so far in the particular cluster or state.

The ability to mathematically build prior knowledge into a reinforcement-learning agent as shown is compelling since the algorithm itself need not change; only the prior is different.

The **BOSS** and **BFS3** algorithms were both run in *Texture World*, and the results are shown in Figure 6.12 and in Figure 6.13. Here, **BOSS** was able to quickly and effectively find the optimal policy, while **BFS3** was sluggish, computationally, and was unable to find the optimal policy.

The parameters for **BOSS** were a first guess that, when run, found an easily identifiable optimal policy, choosing a path that used cells with a high success rate. The parameters for **BFS3** were increased until the resulting policy value appeared to not get closer to the value of **BOSS**'s policy.

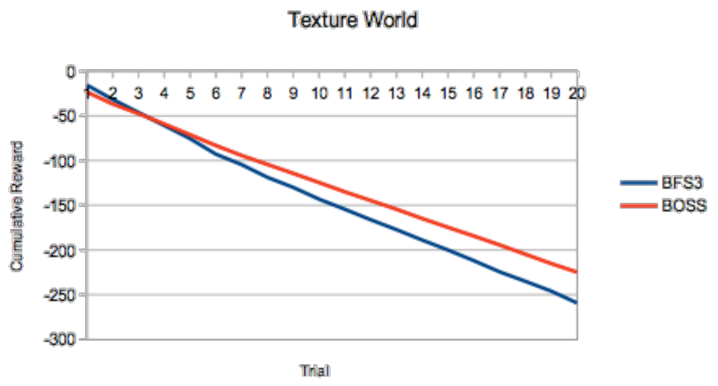

Figure 6.12: *Texture World* on a 5x5 grid. **BOSS** was run with $K = 5$ and $B = 10$. **BFS3** was run with a search depth of $D = 10$, search breadth of $C = 5$, and at most 1500 trajectories per step. In this instance, the actual world had textured set to "true", and the red, green and blue state's chances of success were $0.2, 0.3, 0.8$, respectively. The agents were only given the coloring of each state and the prior; they had to decide for themselves whether or not textures mattered.

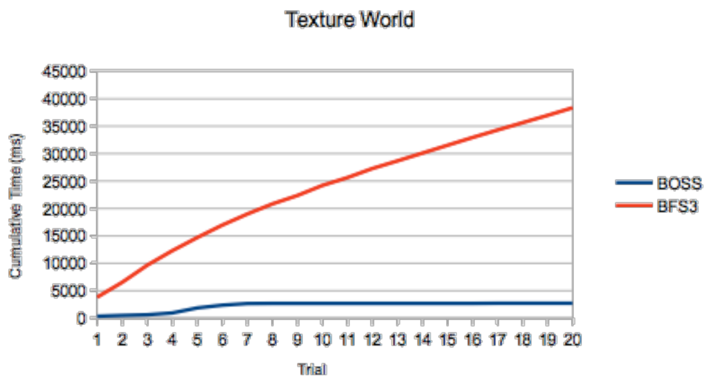

Figure 6.13: *Texture World* on a 5x5 grid. **BOSS**, because it was able to use value iteration for planning in the small world, was very efficient with its computation when compared to **BFS3**, which by definition uses the conservative tree-search planner **FSSS**.

Chapter 7
Conclusion

7.1 Motivation

This section revisits the motivation behind the Bayesian approach to model-based reinforcement learning and why generalized priors are an important aspect of that approach.

One goal of Bayesian approaches to machine learning is to make computer learners more "human" in the predictions they make. Humans are very good at knowledge transfer, or the application of observations made in the past to predictions that need to be made in the future. The use of a prior is an effective and principled way to bring in knowledge from some other domain (or from an algorithm designer's imagination).

The distinction between prior knowledge and algorithm is one that is important and natural, and allows the agent designer to narrow his or her focus. The ability to develop a prior (and the associated inference mechanism) is sufficient for bestowing an agent with some sort of intrinsic knowledge, and this prior can then be plugged directly into **Bayesian DP**, **BOSS**, **BFS3**, or any other algorithm built in this fashion.

Considering the inference separately from the decision making is also advantageous in that there exists a large and active community focused on Bayesian model building and inference. In fact, Latent Dirichlet Allocation [33] provided the original motivation and inspiration for the Cluster and other related CRP-based priors.

7.2 Contributions

This section lists the specific contributions of this dissertation.

7.2.1 Bayesian Modeling

While it is not unprecedented for a reinforcement-learning algorithm to accept flexible priors [20, 22, 35], specific priors that can be adapted to different situations are less common. Chapter 2 discusses several structured priors that can be used for reinforcement-learning algorithms, and discusses in detail the analysis inference methods needed to work with them effectively.

Of special note are the non-parametric models, which allow a reinforcement learner to infer complex structure from observations in ways that are impossible to guess before-hand. The cluster prior was presented previously [1], and the ROAR prior, which uses a CRP to do inference about clustering in continuous spaces, exists only in this dissertation.

Also useful are some of the highly-specialized models mentioned in the experiments in Chapter 6. While they, in particular, are not much use outside of the experiments they are tied to, they serve as an example of how an agent designer can craft a special prior for a special environment. This creation allows the designer to say exactly what parts of the environment are uncertain, and encode the known structure in an efficient and principled way. And, once such a prior is constructed, there exist several algorithms, including two introduced in this dissertation, that can make use of this prior directly.

7.2.2 The BOSS Algorithm

The **BOSS** algorithm and analysis, from Chapter 4, is a major contribution of this dissertation. This algorithm is a flexible and powerful approach for making use of arbitrary priors.

Uncertainly is typically one of the hardest things to quantify in machine learning. Some approaches, like **MBIE** [46] or **RMAX** [3], deal with uncertainty by artificially modifying the reward function of the estimated MDP. **BOSS** addresses uncertainty directly by using posterior samples, which naturally have a correlation between variance and uncertainty. Additionally, **BOSS** is not significantly more computationally expensive, aside from the posterior sampling, than the simplest of model-based learners, since it can use off-the-shelf planning techniques.

Beyond the use of flexible priors, **BOSS** is one of a small collection [2, 9, 21] of Bayesian algorithms that also has theoretical guarantees about its sample complexity. Depending on the particulars of the environment and the prior, **BOSS** can be both PAC-MDP and PAC-BAMDP. This dissertation presents the detailed analysis of these guarantees with respect to **BOSS**, and provides clues about how to make similar analyses for other Bayesian model-based algorithms.

7.2.3 The BFS3 Algorithm

The **BFS3** algorithm and its corresponding analysis is also a major contribution of this dissertation. The approach used by **BFS3** differs from **BOSS** in several important respects. First, it is not planner-agnostic—the analysis depends on **FSSS** [37] being used. Second, instead of sampling MDPs from the posterior, it samples observations directly. Depending on the situation, MDP sampling or observation sampling might be more appropriate, and these two algorithms show that both are possible and practical.

Also, **BFS3** has an attribute that **BOSS** does not have: more is never worse, and often better. That is, raising the values of its parameters will never cause **BFS3** to be less Bayes-optimal. There is some minimum value for the parameters that **BFS3** needs to function correctly, but beyond that it will put anything you give it to good use. Contrast this property with **BOSS**'s sample count parameter, where a value that is too high or too low will render **BOSS** useless.

This algorithm also has theoretical gaurantees. Specifically, it is PAC-BAMDP. Since raising its parameters will eventually cause it to be exactly Bayes-optimal, in the limit, it cannot have a PAC-MDP guarantee, since there are priors that induce a Bayes-optimal policy that is sub-optimal [47].

7.2.4 Separating BOSS and BFS3

The **BOSS** and **BFS3** algorithms are distinct in the way they work with posteriors. **BOSS** requires a posterior for sampling MDPs, while **BFS3** operates directly on experience. The type of model that is appropriate for a specific experiment is very context-dependent. It could be that it is difficult to sample the entire model at once. There are tricks, especially the trick used for sampling from ROAR in Section 2.3.2, that allow **BOSS** to treat an experience posterior as an approximate model posterior, but they can add complication to the inference process. On the other hand, for **BFS3** to effectively use an experience sampler, it needs to be able to quickly sample from the posterior conditioned on any possible history, since its search tree will contain belief-states with many different histories. The amount of overhead required by the

ROAR model makes it infeasible to use with **BFS3**.

Beyond the different inference processes used by **BOSS** and **BFS3**, they also have very different computational requirements. Since **BOSS** is planner-agnostic—**BOSS** can make use of any planner that can work with its hyper-models—it can often take advantage of specifics of the domain to plan more efficiently. For instance, with smaller discrete domains, it can quickly plan using value iteration. The **BFS3** algorithm is tied specifically to tree search. Since its goal is to approach Bayes-optimality, it searches the BAMDP, and the problem is generally intractable. However, **FSSS** can often be an effective planner in practice, as shown in Section 6.3 and previous work [37]. In those situations, **BFS3** is a good algorithm choice for deciding how to explore the belief-state-space in order to get to Bayes-optimality as quickly as possible.

7.3 Concluding Remarks

Encoding prior knowledge into a reinforcement-learning algorithm is hard. Using the Bayesian approach, where the knowledge is encoded into a prior distribution, allows a researcher to compartmentalize some of the difficulty. Inferring structure in environments is especially difficult to do without using the Bayesian approach. This dissertation aims to address two important aspects of Bayesian model-based reinforcement learning: the construction and inference of priors and posteriors, and the algorithms that make effective use of those priors and posteriors.

Bibliography

[1] John Asmuth, Lihong Li, Michael L. Littman, Ali Nouri, and David Wingate. A bayesian sampling approach to exploration in reinforcement learning. In *Proceedings of The 25th Conference on Uncertainty in Artificial Intelligence (UAI-09)*. 2009. URL `http://paul.rutgers.edu/~jasmuth/pub/uai09-boss.pdf`. iv, xiv, 79, 86, 127, 151

[2] John Asmuth and Michael Littman. Learning is planning: near bayes-optimal reinforcement learning via monte-carlo tree search. In *Proceedings of The 27th Conference on Uncertainty in Artificial Intelligence (UAI-11)*. 2011. URL `http://paul.rutgers.edu/~jasmuth/bayes/asmuth11.pdf`. iv, 152

[3] Ronen I. Brafman and Moshe Tennenholtz. R-MAX—A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002. xix, 10, 122, 141, 152

[4] Nicholas K. Jong and Peter Stone. Model-based exploration in continuous state spaces. In *The Seventh Symposium on Abstraction, Reformulation, and Approximation*. 2007. 8, 62, 64, 122

[5] Bethany R. Leffler, Michael L. Littman, and Timothy Edmunds. Efficient reinforcement learning with relocatable action models. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*. 2007. URL `http://www.research.rutgers.edu/~tedmunds/publications/RAM-aaai2007.pdf`. 8, 33, 38, 127

[6] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992. 9

[7] Michael Duff. Design for an optimal probe. In *Proceedings of the 20th International Conference on Machine Learning*. 2003. 11, 65

[8] J. Zico Kolter and Andrew Y. Ng. Near-Bayesian exploration in polynomial time. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 513–520. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-516-1. doi: http://doi.acm.org/10.1145/1553374.1553441. URL `http://doi.acm.org/10.1145/1553374.1553441`. 13, 62, 73

[9] M. Araya, O. Buffet, and V. Thomas. Near-optimal BRL using optimistic local transitions. In *Proceedings of the 29th International Conference on Machine Learning*, pages 97–104. 2012. 13, 62, 75, 152

[10] Lihong Li. *A Unifying Framework for Computational Reinforcement Learning Theory (pp 78-79)*. Ph.D. thesis, Rutgers University, 2009. 16, 96, 113

[11] Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2–3):209–232, 2002. 16, 18

[12] Lihong Li. *A Unifying Framework for Computational Reinforcement Learning Theory (pp 163-164)*. Ph.D. thesis, Rutgers University, 2009. 18, 96

[13] Y.W. Teh. Dirichlet process. *Encyclopedia of machine learning*, pages 280–287, 2010. 26

[14] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I. Jordan. An Introduction to MCMC for Machine Learning. In *Machine Learning*, pages 5–43. Kluwer Academic Publishers, 2003. 28

[15] Radford M. Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, Vol. 9(No. 2):pp. 249–265, June 2000. 28

[16] Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian Q-learning. In *Fifteenth National Conference on Artificial Intelligence (AAAI)*, pages 761–768. 1998. URL `http://www.cs.huji.ac.il/~nirf/Papers/DFR1.pdf`. 29

[17] Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 697–704. 2006. 29, 72, 126

[18] Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 956–963. ACM, New York, NY, USA, 2005. ISBN 1-59593-180-5. doi: http://doi.acm.org/10.1145/1102351.1102472. 30, 62, 78, 79, 106

[19] Carlos Diuk, Lihong Li, and Bethany R Leffler. The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 249–256. ACM, 2009. 33

[20] Malcolm J. A. Strens. A Bayesian framework for reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 943–950. 2000. 34, 62, 67, 72, 124, 126, 127, 151

[21] Jonathon Sorg, Satinder Singh, and Richard L. Lewis. Variance-based rewards for approximate Bayesian reinforcement learning. In *Proceedings of the 26th Conference on Uncertainty in Artifical Intelligence (UAI-10)*. 2010. 34, 62, 74, 145, 152

[22] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: A hierarchical Bayesian approach. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007)*, pages 1015–1022. 2007. 34, 71, 151

[23] D. Fink. A compendium of conjugate priors. *See http://www. people. cornell. edu/pages/df36/CONJINTRnew% 20TEX. pdf*, page 46, 1997. 40

[24] C.E. Rasmussen. The infinite gaussian mixture model. *Advances in neural information processing systems*, 12(5.2):2, 2000. 47

[25] Ronen I. Brafman and Moshe Tennenholtz. Efficient learning equilibrium. In *Advances in Neural Information Processing Systems*, volume 9, pages 1635–1643. 2003. 62, 86

[26] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. Approximate planning in large POMDPs via reusable trajectories, 1999. Submitted. 62, 75

[27] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML-06)*, volume 4212, pages 282–293. Springer Berlin / Heidelberg, 2006. URL `http://dx.doi.org/10.1007/11871842_29`. 62, 81, 106, 110

[28] Ali Nouri and Michael L. Littman. Multi-resolution exploration in continuous spaces. In *Proceedings of Neural Information Processing Systems*. 2009. 62

[29] John Asmuth, Michael L. Littman, and Robert Zinkov. Potential-based shaping in model-based reinforcement learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*. 2008. URL `http://paul.rutgers.edu/~jasmuth/pub/aaai08-shaping.pdf`. 63, 122

[30] S.J. Russell, P. Norvig, J.F. Canny, J.M. Malik, and D.D. Edwards. *Artificial Intelligence: a Modern Approach*, volume 2. Prentice hall Englewood Cliffs, NJ, 1995. 64

[31] Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. PAC model-free reinforcement learning. In *Proceedings of the Twenty-third International Conference on Machine Learning (ICML-06)*. 2006. URL `http://www.cs.rutgers.edu/~strehl/papers/ICML06PACModelFreeRL.pdf`. 74

[32] William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285–294, 1933. 78

[33] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003. 79, 150

[34] David M. Blei, Thomas L. Griffiths, Michael I. Jordan, and Joshua B. Tenenbaum. Hierarchical topic models and the nested chinese restaurant process. In *Advances in Neural Information Processing Systems*, page 2003. MIT Press, 2004. 79

[35] Finale Doshi. The infinite partially observable markov decision process. In *Neural Information Processing Systems*, volume 22, pages 477–485. 2009. 79, 151

[36] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, May 2002. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1013689704352. URL `http://dx.doi.org/10.1023/A:1013689704352`. 81

[37] Thomas Walsh, Sergiu Goschin, and Michael Littman. Integrating sample-based planning and model-based reinforcement learning. In *Proceedings of the Association for the Advancement of Artificial Intelligence*. 2010. 83, 106, 152, 154

[38] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. In *Proceedings of the 15th International Conference on Machine Learning*, pages 260–268. 1998. URL `citeseer.nj.nec.com/kearns98nearoptimal.html`. 96

[39] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1324–1331. 1999. 105

[40] Sham M. Kakade. *On the Sample Complexity of Reinforcement Learning*. Ph.D. thesis, Gatsby Computational Neuroscience Unit, University College London, 2003. 113

[41] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1994. ISBN 0-13-103805-2. 131, 145

[42] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 369–376. The MIT Press, Cambridge, MA, 1995. 134

[43] J. C. Gittins. *Multi-armed Bandit Allocation Indices*. Wiley-Interscience series in systems and optimization. Wiley, Chichester, NY, 1989. 140

[44] S. N. MacEachern and P. Muller. Estimating mixture of Dirichlet process models. *Journal of Computational and Graphical Statistics*, 7(2):223–238, June 1998. 140

[45] Thomas J. Walsh, István Szita, Carlos Diuk, and Michael L. Littman. Exploring compact reinforcement-learning representations with linear regression. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 591–598. AUAI Press, Arlington, Virginia, United States, 2009. ISBN 978-0-9749039-5-8. URL `http://portal.acm.org/citation.cfm?id=1795114.1795183`. 142

[46] Alexander L. Strehl and Michael L. Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the Twenty-second International Conference on Machine Learning (ICML-05)*, pages 857–864. 2005. URL `http://paul.rutgers.edu/~strehl/papers/icml05-mbie.pdf`. 152

[47] Lihong Li. *A Unifying Framework for Computational Reinforcement Learning Theory (pp 80-82)*. Ph.D. thesis, Rutgers University, 2009. 153

Vita

John Thomas Asmuth

1999–2004	B. Sc. in Computer Science from Rutgers University
2011	Intern, Sarnoff Corporation
2010	Intern, AT&T Research
2006–2012	Teaching Assistant, Rutgers University
2009–2012	Graduate Assistant, Rutgers University

Publications

John Asmuth and Michael Littman. Learning is planning: near Bayes-optimal reinforcement learning via Monte-Carlo tree search. In *Proceedings of The 27th Conference on Uncertainty in Artificial Intelligence (UAI-11)*. 2011. URL `http://paul.rutgers.edu/~jasmuth/bayes/asmuth11.pdf`.

John Asmuth and Michael Littman. Appendix. Technical Report DCS-tr-687, Rutgers University department of Computer Science, 2011.

John Asmuth and ML Littman. Approaching Bayes-optimalilty using Monte-Carlo tree search. In *Proc. 21st Int. Conf. Automat. Plan. Sched., Freiburg, Germany*. 2011.

John Asmuth, Lihong Li, Michael L. Littman, Ali Nouri, and David Wingate. A Bayesian Sampling Approach to Exploration in Reinforcement Learning. In *Proceedings of The 25th Conference on Uncertainty in Artificial Intelligence (UAI-09)*. 2009. URL `http://paul.rutgers.edu/~jasmuth/pub/uai09-boss.pdf`.

John Asmuth, Michael L. Littman, and Robert Zinkov. Potential-based Shaping in Model-based Reinforcement Learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*. 2008. URL `http://paul.rutgers.edu/~jasmuth/pub/aaai08-shaping.pdf`.

Håkan L. S. Younes, Michael L. Littman, David Weissman, and John Asmuth. The First Probabilistic Track of the International Planning Competition. *Journal of Artificial Intelligence Research*, 24:851–887, 2005.