Towards Global Reinforcement Learning

by

Milen Pavlov

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Sequential decision making under uncertainty is a ubiquitous problem. In everyday situations we are faced with a series of decisions that aim to maximize the probability of achieving some goal. However, decision outcomes are often uncertain and it is not always immediately evident how to determine if one decision is better than another. The Reinforcement Learning framework overcomes this difficulty by learning to make optimal decisions based on interactions with the environment. One drawback of Reinforcement Learning is that it requires too much data (interactions) to learn from scratch. For this reason, current approaches attempt to incorporate prior information in order to simplify the learning process. However, this is usually accomplished by making problem-specific assumptions, which limit generalizability of the approaches to other problems. This thesis presents the first steps towards a new framework that incorporates and exploits broad prior knowledge in a principled way. It uses Constraint Satisfaction and Bayesian techniques to construct and update a belief over the environment, as well as over good decisions. This allows for incorporating broad types of prior knowledge without limiting generalizability. Preliminary experiments show that the framework's algorithms work well on toy problems in simulation and encourage further research on real-world problems.

Contents

List of Figures

Chapter 1

Introduction

Sequential decision making is a ubiquitous problem. In everyday situations, such as grocery shopping, driving a car, playing competitive sports or even just socializing with friends, we are faced with a series of decisions. Should I buy some broccoli or go for spinach instead? Should I take the next exit or stay on and hope the traffic jam clears up? Should I pass the ball or try to score myself? In such situations we often have a specific goal in mind (e.g., cooking a delicious and healthy meal) and we aim to make decisions that maximize the probability of achieving this goal (e.g., buying the appropriate vegetables).

Sequential decision making is not too difficult to reason about if we can predict with certainty the effects of all our decisions. Unfortunately, this is often not the case. Many times we need to make decisions without fully understanding their effects. For example, we cannot tell if passing to a teammate will result in winning the game. We do not know if taking the next exit will not result in getting stuck in a worse traffic jam. In such scenarios we are faced with making decisions *under uncertainty* and it is not immediately evident how to determine if one decision is better than another.

Fortunately, in Operations Research there already exists a framework for sequential decision making under uncertainty. This framework is referred to as a Markov Decision Process. A Markov Decision Process models the world as a finite state machine: it assumes that the world can be in one of a set of predefined states and that decisions affect the world by changing its current state to another. This change is referred to as a *transition*. The uncertainty in the effects of decisions is modelled by means of a stochastic transition function. That is, from any state, any decision has some specific probability to result in a transition to any other state. States and decisions are also associated with some *rewards* (utilities). For example, being in one state might be more rewarding than being in another. Thus, by

combining probability theory and utility theory, we can reason about the expected reward of each possible decision. From there, the task is simple: simply choose decisions that maximize the expected reward.

For example, consider the game of football (or soccer, as it is known in North America). The game starts with score 0-0. This can be viewed as the state of the world and we will refer to it as state "00". If the away team scores then the score becomes 0-1 and the world transitions to state "01". Suppose we were on the away team and would understandably associate a high reward with such a transition. Coming back to the decision of passing, we could model the problem by associating specific probabilities with the different outcomes of different decisions. Given the choice between passing to Cristiano Ronaldo or passing to Wayne Rooney, it is clear that either decision has a nonzero (and likely much higher) probability of transitioning to state "01". Of course, given that Rooney is a skilled striker, the probability associated with passing to him might be slightly higher. Therefore, the expected reward associated with the decision to pass to him will also be higher and so this will be the optimal decision to make.

But what if the transitions and rewards were not known in advance? What if we did not know who Rooney was and really could not tell who had the better chance of scoring? What if, in addition to the reward associated with scoring, we also associated another reward with just watching Ronaldo play attractive football, and did not know *a priori* which of the two rewards was greater?

This is where Reinforcement Learning becomes useful. Reinforcement Learning allows sequential decision making under uncertainty even when the transitions and rewards are unknown. It works by observing the feedback of our decisions (particularly, the resulting transition and the associated reward) and learning to expect a similar feedback the next time we find ourselves in the same situation. In this way, after sufficient decision making, the transition function and the reward function are both learned.

However, Reinforcement Learning is not easy to perform. Out of the three Machine Learning frameworks—supervised, unsupervised and reinforcement—it is perhaps the most difficult, due to the following challenges:

1. Feedback is received in terms of reinforcement (e.g., a reward of "5") instead of labeled data (e.g., "this action was the correct one"). This makes it difficult to reason about the optimality of our decisions. For example, sliding on the grass to steal the ball from an opponent might result in a success and some positive reinforcement, but from that evidence alone it is not clear whether sliding is always the best option.

2. The sequential aspect implies that the reward for a decision might come very late, perhaps after making numerous other decisions along the way. This makes it difficult to properly assign credit to individual decisions. For example, if Rooney scores a goal, is the credit his for delivering a piercing shot in the top corner of the net, or does Ronaldo also deserve a substantial part of it for whirling past three defenders, drawing the goalkeeper out and delivering the perfect pass?

3. Data (in terms of reinforcements) is often very limited and/or costly because it is obtained through exploration (i.e., interacting with the environment). Complete exploration is generally impossible for two reasons. First, the amount of time required to explore all possible effects of all possible decisions is often prohibitively large. For example, no football player has ever tried all possible ways of scoring a goal. And second, there usually exist decisions that can potentially incur very large penalties (negative reinforcement) and should never be taken. For example, most football players would never decide to vent frustration by headbutting an opponent, as it might result in a red card and a ban from the sport.

The first two challenges are well documented and explored in existing research on Reinforcement Learning. This thesis focuses on the last one.

Prior information helps immensely with this challenge. If someone told us in advance that specific decisions would lead to specific outcomes then (barring any trust issues) it would greatly simplify our learning process. Let us see why this is the case. In the beginning of the learning process we know nothing about anything: the real world could be one of infinitely many hypothetical worlds. To learn, we observe the feedback of our decisions (and the ensuing actions) and start to favour hypotheses that are consistent with this feedback. Prior knowledge helps by readily specifying which hypotheses are favourable. It saves us all the exploration necessary to determine this ourselves and so greatly speeds up the learning process. For this reason, there is great motivation to incorporate as much prior information as possible into learning problems.

Unfortunately, to our knowledge there does not currently exist any framework for incorporating such broad prior knowledge in Reinforcement Learning, nor developing general algorithms that make use of it. The state of the art consists of designing algorithms that employ ad-hoc shortcuts, which often rely on problem-specific assumptions. As a result, it is hard to generalize these approaches to many problems. Moreover, prior information is often only used implicitly through those assumptions, which introduces difficulties with validating its correctness.

This thesis makes the first step to providing a framework to encode prior knowledge in a more general and principled way. We consider a Bayesian approach, whereby prior knowledge can be modelled as a probability distribution over hypotheses. This approach provides a principled way to make decisions based on that distribution and also to update the distribution as a result of observing decisions' outcomes. The main problem is how to construct such a prior disribution so that it incorporates prior information about transitions, rewards *and* courses of actions (policies). Doing this would enable us not only to exploit one or two pieces of information to learn about the third, but also to exploit all three to learn about all three. In this document, we explore how this problem can be solved using various Constraint Satisfaction techniques.

We summarize our contributions as follows:

- *Incorporating prior knowledge over both transition models of the environment and optimal policies.* The benefit of this is twofold. One, prior knowledge reduces uncertainty in both the true transition model and the optimal policy, which helps increase the rate of learning. And two, incorporating prior knowledge about policies avoids performing potentially costly actions.

- *Constructing a joint prior.* This enables learning agents to simultaneously reason about all pieces of prior knowledge given. We present two construction approaches: one that estimates a joint distribution directly and one that instead represents its constituent single and conditional distributions.

- *Optimizing action selection directly.* One of the approaches mentioned in the previous point achieves this by sampling a policy from the marginal of the joint distribution; the other - by using an already existing learning algorithm (BEETLE [14]). In both cases, selected actions tend to be more exploratory in the beginning of the learning process and more rewarding as time progresses.

- *Proposing a measure of inconsistency between the two types of prior knowledge.* This may help identify problems with the elicited prior knowledge and suggest if any of it should be rejected to preserve consistency.

This thesis is organized as follows. Chapter 2 reviews Reinforcement Learning, Bayesian learning and other algorithms that provide the building blocks for our research. Chapter 3 offers a brief survey of current state-of-the-art learning approaches. Chapter 4 details the framework we propose. Chapter 5 shows our experiments and discusses their results. Chapter 6 concludes and suggests directions for further research. Appendix 6.1 summarizes all symbols defined or otherwise referred to in the document.

Chapter 2

Background

This chapter explains notation and concepts used throughout the rest of the document. It is organized as follows. First, an overview of the Bayesian approach to learning is given. This includes a discussion of useful ways of representing probability distributions in the context of learning. Next, we review the Reinforcement Learning framework and how it can be used to learn the parameters of a Markov Decision Process. Finally, we compare two methods for satisfying linear constraints. While these are not generally related to the theory of Learning, in our case they are useful for approximating learning parameters.

2.1 Bayesian Learning

Discussions on probabilities and statistics are often classified into one of two views: the Classical and the Bayesian. The Classical view states that the probability of an event x is equal to the expected relative frequency of occurrence of x. That is, if we denote by x_n the outcome of experiment n, then the *physical probability* of outcome x is defined as

$$p(x) = \lim_{n\to\infty} \frac{\sum_{i=1}^{n} \delta(x_i = x)}{n}$$

$$\delta(cond) = \begin{cases} 1 & \text{if } cond \text{ evaluates to } true \\ 0 & \text{otherwise} \end{cases}$$

Using the above formula, classical statisticians are able to estimate the likelihood of event x occurring in the n-th experiment, given the outcomes of the previous $n-1$ experiments.

In contrast, the Bayesian view states that the probability of an event is equal to the *degree of belief* someone assigns to it. Thus, it is not a physical property of the environment, but rather a person's belief in it. Often, such beliefs are given by experts who have extensive knowledge of the environment and thus the ability to give accurate estimations.

In the rest of this document, *Bayesian probabilities* will be referred to simply as *probabilities*. In case we explicitly need to refer to a probability in the classical sense, we will use the term *physical probability*.

First, let us review the standard notation used in Probability and Statistics. We denote variables with uppercase letters (e.g., X, Θ_i) and values with their corresponding lowercase letters (e.g., x, θ_i). It is sometimes useful to talk about sets or vectors of variables. These we denote with bold uppercase letters (e.g., $\boldsymbol{X}$, $\boldsymbol{\Theta_i}$). Similarly, a set of values we denote by the corresponding bold lowercase letters (e.g., $\boldsymbol{x_i}$, $\boldsymbol{\theta_i}$). We use $p(X = x \mid \xi)$ to denote one's degree of belief that $X = x$, based on one's state of information ξ. With $p(x \mid \xi)$ we denote the probability distribution for X. Whenever ambiguity does not arise, we may also use $p(x \mid \xi)$ as a shorthand notation for $p(X = x \mid \xi)$.

Now we can proceed to define the Bayesian approach to learning. We use the convention of Heckerman [7]. Consider a variable X, whose probability distribution we wish to learn. Let us denote this unknown probability distribution by Θ. Consider also a set of observations (sometimes referred to as *evidence*), $\boldsymbol{e} = \{X_1 = x_1, ..., X_N = x_N\}$, where X_i denotes the variable representing the outcome of the i-th experiment and x_i - the outcome itself. The goal is to estimate the likelihood that a certain outcome will occur in the $(N + 1)$-th experiment. That is, we need to compute $p(x_{N+1} \mid \boldsymbol{e}, \xi)$. Bayesians do this by keeping an updated distribution over Θ, $p(\theta \mid \boldsymbol{e}, \xi)$, and using it to estimate $p(x_{N+1} \mid \boldsymbol{e}, \xi)$. Let us see how we can use Bayes' Rule to keep updating $p(\theta \mid \xi)$ every time we receive new evidence $\boldsymbol{e}$:

$$p(\theta \mid \boldsymbol{e}, \xi) = \frac{p(\theta \mid \xi) p(\boldsymbol{e} \mid \theta, \xi)}{p(\boldsymbol{e} \mid \xi)} \tag{2.1}$$

To see how we can perform this computation let us examine each of the terms in the equation. The denominator is a normalization constant and can be computed using:

$$p(\boldsymbol{e} \mid \xi) = \int_{\theta} p(\theta \mid \xi) p(\boldsymbol{e} \mid \theta, \xi) d\theta$$

The second term in the quotient is readily computable, since the probability of each event in $\boldsymbol{e}$ is dictated by the given θ. Finally, the first term in the quotient is assumed known in advance. Normally, this is either through expert estimation or

through some other evidence occurring before $\boldsymbol{e}$. (More discussion on this is given later in this section.)

This method of computing a *posterior* distribution $p(\theta \mid \boldsymbol{e}, \xi)$ from a *prior* distribution $p(\theta \mid \xi)$ by incorporating some observed evidence $\boldsymbol{e}$ is referred to as a Bayesian update.

Once we have computed the posterior, we can compute our estimate of the probability of any one event x_{N+1} occurring in the next experiment.

$$p(x_{N+1} \mid \boldsymbol{e}, \xi) = \int_{\theta} p(x_{N+1} \mid \theta, \xi) p(\theta \mid \boldsymbol{e}, \xi) d\theta$$

There is one more detail that should be pointed out. Before we start learning, we need a prior distribution that reflects our *a priori* knowledge of the problem domain. There are two challenges associated with constructing such a prior. One, it must be able to represent our prior knowledge relatively well. And two, it should facilitate computation. For example, a *conjugate* prior is, by definition, computable in closed form under the Bayesian update of (2.1). This allows the same representation to be used for the posterior distribution so that it is easily computable using dynamic programming techniques.

For very simple problem domains where the unknown variable is binary (e.g., with possible values 0 and 1) we only need to learn a single distribution (e.g., θ_0, the probability of observing a zero). Then, the Beta distribution offers a good model for representation.

$$p(\theta_0 \mid \xi) = \text{Beta}(\theta_0; \alpha_0, \alpha_1) \equiv \frac{1}{B(\alpha_0, \alpha_1)} (\theta_0)^{\alpha_0 - 1} (1 - \theta_0)^{\alpha_1 - 1} \tag{2.2}$$

where the normalization constant $1/B(\alpha_0, \alpha_1)$ is given by the Beta function:

$$B(\alpha_0, \alpha_1) = \int_0^1 (t)^{\alpha_0 - 1} (1 - t)^{\alpha_1 - 1} dt = \frac{\Gamma(\alpha_0)\Gamma(\alpha_1)}{\Gamma(\alpha_0 + \alpha_1)}$$

Here $\Gamma()$ is the gamma function - a generalization of the factorial function to the domain of real numbers. In this scenario, the Beta distribution model is a good choice because of the following two points. One, it can readily represent our prior knowledge. Specifically, if we believe that, say, seven times out of ten we would observe a zero then we can set $\alpha_0 = 7$ and $\alpha_1 = 10 - 7 = 3$. And two, its posterior is computable in closed form. This is clear from the following. For evidence $e =$

$\{X_i = 0\}$, Bayes' Rule (2.1) gives us

$$\begin{aligned} p(\theta_0 \mid e, \xi) &= \frac{p(\theta_0 \mid \xi) p(e \mid \theta_0, \xi)}{p(e \mid \xi)} \\ &= \frac{\text{Beta}(\theta_0; \alpha_0, \alpha_1) p(e \mid \theta_0, \xi)}{p(e \mid \xi)} \\ &= k(\theta_0)^{\alpha_0 - 1} (1 - \theta_0)^{\alpha_1 - 1} p(X_i = 0 \mid \theta_0, \xi) \\ &= k(\theta_0)^{\alpha_0 - 1} (1 - \theta_0)^{\alpha_1 - 1} \theta_0 \\ &= k(\theta_0)^{\alpha_0 - 1 + 1} (1 - \theta_0)^{\alpha_1} \\ &= \text{Beta}(\theta_0; \alpha_0 + 1, \alpha_1) \end{aligned}$$

where the normalization constant k is computed by

$$k = \frac{1}{B(\alpha_0, \alpha_1) p(e \mid \xi)}$$

As we can see from above, the posterior is still a Beta distribution. In fact, the only changes from the prior are an increment of the hyperparameter corresponding to the observed event and an update of the normalization constant. The procedure is symmetric for evidence $e = \{X_i = 1\}$.

However, in other scenarios it is desirable to track multi-valued variables and in those cases we need other models of representation. We consider two such models: The Dirichlet distribution and the mixture of Dirichlet distributions. The two are discussed in detail in the following two subsections respectively.

2.1.1 The Dirichlet Distribution

The Dirichlet distribution is the multivariate generalization of the Beta distribution [6]. For some set of parameters $\boldsymbol{\theta} = \{\theta_1, ..., \theta_I\}$ and a set of corresponding hyperparameters $\boldsymbol{\alpha} = \{\alpha_1, ..., \alpha_I\}$, the probability distribution of $\boldsymbol{\theta}$ is given by

$$\text{Dir}(\boldsymbol{\theta}; \boldsymbol{\alpha}) \equiv \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{I} (\theta_i)^{\alpha_i - 1} \tag{2.3}$$

where the normalization constant $1/B(\boldsymbol{\alpha})$ is given by the multinomial Beta function

$$B(\boldsymbol{\alpha}) = \int_{\boldsymbol{\theta}} \prod_{i=1}^{I} (\theta_i)^{\alpha_i - 1} = \frac{\prod_i \Gamma(\alpha_i)}{\Gamma(\sum_i \alpha_i)} \tag{2.4}$$

Similar to the Beta distribution, the Dirichlet has the following constraints on its parameters.

$$\sum_i \theta_i = 1$$

$$\forall i : \alpha_i > 0,\, \theta_i \geq 0$$

The Dirichlet distribution is more expressive than the Beta distribution because it can encode knowledge about multivariate data. For example, if we observe the results of throwing a biased 6-sided die and increment α_i every time we see an i, $1 \leq i \leq 6$, then with enough throws we can estimate the physical bias of the die. (In fact, $\text{Dir}(\boldsymbol{\theta}; \boldsymbol{\alpha})$ gives us a probability distribution over all possible biases θ.)

Like the Beta distribution, the Dirichlet is also closed under a Bayesian update: Given evidence that in the $(n+1)$-th experiment we observed outcome x_j, $e = \{X_{n+1} = x_j\}$, we can compute the posterior as follows.

$$\begin{aligned}
p(\boldsymbol{\theta} \mid e, \xi) &= \frac{p(\boldsymbol{\theta} \mid \xi) p(e \mid \boldsymbol{\theta}, \xi)}{p(e \mid \xi)} \\
&= k \left[\prod_i (\theta_i)^{\alpha_i - 1} \right] p(X_{n+1} = x_j \mid \boldsymbol{\theta}, \xi) \\
&= k \left[\prod_i (\theta_i)^{\alpha_i - 1} \right] \theta_j \\
&= k \prod_i (\theta_i)^{\alpha_i - 1 + \delta(i=j)} \\
&= \text{Dir}(\boldsymbol{\theta}; \boldsymbol{\alpha} + \hat{\boldsymbol{e}}_j) \\
\hat{\boldsymbol{e}}_j &= \text{The j-th component of the } I \times I \text{ identity matrix} \\
k &= \frac{1}{B(\boldsymbol{\alpha}) p(e \mid \xi)}
\end{aligned}$$

Since the posterior is also a Dirichlet distribution (with one of the hyperparameters incremented), this model is suitable for Bayesian learning.

To construct a Dirichlet from prior knowledge we can use similar techniques as with the Beta distribution. Let us define the *precision* s and the *mean* $\boldsymbol{m}$ of the Dirichlet as follows:

$$s = \sum_i \alpha_i$$

$$\boldsymbol{m} = \frac{\boldsymbol{\alpha}}{s}$$

Note that while the precision is a static number, the mean is vector of probabilities, collectively denoting the expectation of the Dirichlet. Given both the precision and mean, constructing the corresponding Dirichlet (i.e., determining the initial values of its hyperparameters) is trivial:

$$\boldsymbol{\alpha} = s\boldsymbol{m}$$

Admittedly, many real-world problems do not naturally lend themselves to estimating means and precision. For lack of a more practical way of constructing Dirichlets however, this is currently the *de facto* standard.

Finally, it is worth noting that the Dirichlet distribution is *unimodal*. That is, it encodes distributions that are peaked towards a single value. In scenarios where we need to encode multimodal distributions we need to use a more expressive model, such as the Dirichlet Mixture.

2.1.2 The Dirichlet Mixture

There are cases in which even the Dirichlet distribution is not expressive enough to model the environment. For those cases we may choose to use a mixture of Dirichlet distributions. We now proceed to define the formula and properties of this Dirichlet Mixture.

For some set of parameters $\boldsymbol{\theta} = \{\theta_1, ..., \theta_I\}$, a second-order set of hyperparameters $\boldsymbol{\alpha} = \{\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_D\}$ (where each $\boldsymbol{\alpha}_d = \{\alpha_{d1}, ..., \alpha_{dI}\}$) and a set of weights $\boldsymbol{c} = \{c_1, ...c_D\}$, the probability of $\boldsymbol{\theta}$ is given by

$$\text{DirMix}(\boldsymbol{\theta}; \boldsymbol{c}, \boldsymbol{\alpha}) \equiv \sum_{d=1}^{D} c_d \text{Dir}(\boldsymbol{\theta}; \boldsymbol{\alpha}_d) = \sum_{d=1}^{D} c_d \frac{1}{B(\boldsymbol{\alpha}_d)} \prod_{i=1}^{I} (\theta_i)^{\alpha_{di}-1} \tag{2.5}$$

The normalizing constants $1/B(\boldsymbol{\alpha}_d)$ are given by the multinomial Beta function, as defined in (2.4). There are also the following constraints on the parameters of the distribution.

$$\sum_i \theta_i = 1$$

$$\sum_d c_d = 1$$

$$\forall d, i: \ \alpha_{di} > 0, \ c_d \geq 0, \ \theta_i \geq 0$$

The Dirichlet Mixture is more expressive than a single Dirichlet because it can model distributions over data drawn from different Dirichlets. For example, we can model word frequencies in a document using a single Dirichlet distribution.[1] This works well for documents that come from the same domain of knowledge (e.g., computer science theses). However, in cases where documents come from a variety of domains, word frequencies in documents from different domains do not follow the same Dirichlet distribution. In those cases, a Dirichlet Mixture is a better fit.

[1]Specifically, θ_i is the probability that any word is word i and α_i is the count of observed occurrences of word i in the document.

If we take a closer look at the analytical forms of the plain Dirichlet and the Dirichlet Mixture, we can readily see why the second is more expressive: it takes the form of a polynomial rather than a monomial. Since each monomial term in the polynomial corresponds to a plain Dirichlet (which is unimodal), the complete mixture can be used to encode multimodal distributions. In fact, it is well known that polynomials can be used to effectively approximate any function. Therefore, any probability distribution could potentially be approximated with a Dirichlet Mixture. The quality of approximation would then depend on the number of monomial terms in the mixture.

We show that a Dirichlet Mixture is also closed under a Bayesian update. Given evidence that in the $(n+1)$-th experiment we observed outcome x_j, $e = \{X_{n+1} = x_j\}$, we can compute the posterior as follows.

$$
\begin{aligned}
p(\boldsymbol{\theta} \mid e, \xi) &= \frac{p(\boldsymbol{\theta} \mid \xi) p(e \mid \boldsymbol{\theta}, \xi)}{p(e \mid \xi)} \\
&= \frac{1}{p(e \mid \xi)} \left[\sum_d c_d \frac{1}{B(\boldsymbol{\alpha}_d)} \prod_i (\theta_i)^{\alpha_{di}-1} \right] \theta_j \\
&= \sum_d \frac{c_d}{p(e \mid \xi)} \cdot \frac{B(\boldsymbol{\alpha}_d + \hat{\boldsymbol{e}})}{B(\boldsymbol{\alpha}_d)} \cdot \frac{1}{B(\boldsymbol{\alpha}_d + \hat{\boldsymbol{e}})} \prod_i (\theta_i)^{\alpha_{di}-1+\delta(i=j)} \\
&= \sum_d c'_d \mathrm{Dir}(\boldsymbol{\theta}; \boldsymbol{\alpha}_d + \hat{\boldsymbol{e}}) \\
&= \mathrm{DirMix}\left(\boldsymbol{\theta}; \boldsymbol{c'}, \bigcup_d \{\boldsymbol{\alpha}_d + \hat{\boldsymbol{e}}_j\} \right) \\
c'_d &= \frac{B(\boldsymbol{\alpha}_d + \hat{\boldsymbol{e}})}{p(e \mid \xi) B(\boldsymbol{\alpha}_d)} c_d
\end{aligned}
$$

Constructing a Dirichlet Mixture is not as straightforward as other previously discussed distributions. The method of specifying mean and precision, while sound in theory, proves difficult to apply in practice. This is because, in the case of mixtures, several (mean,precision)-tuples need to be specified, in addition to a set of weight values. Learning problems usually do not possess structure that could intuitively be converted to this format. Furthermore, we are not aware of any other principled methods for constructing a Dirichlet Mixture.

2.2 Reinforcement Learning

While Bayesian Learning deals with inferring probability distributions over unknown variables, which could potentially be used as a basis for making good decisions in different situations, Reinforcement Learning deals with the problem of

making good decisions directly. By adding the notion of an *agent* to the environment and explicitly specifying possible *actions*, Reinforcement Learning aims to learn which actions the agent should perform in which states of the environment. Let us see how this problem can be formulated mathematically.

The *environment* in Reinforcement Learning settings is usually modelled using a *Markov Decision Process* (*MDP*) [1]. An MDP is formally defined as a tuple $\langle S, A, T(), R() \rangle$, where:

- S is the set of *states* of the environment (also referred to as the *state space*). We will denote states with $s \in S$.
- A is the set of *actions* that agents can perform in any of the states. We will denote actions with $a \in A$.
- $T : S \times A \times S \rightarrow [0, 1]$ is a nondeterministic *transition function* that gives the likelihood that the environment will transition to some specific state as a result of some action being executed in some other state. Therefore, $T(s, a, s') \equiv p(s' \mid s, a)$. Note that T may be partially or completely unknown to the agent.
- $R : S \times A \rightarrow \mathbb{R}$ is a *reward function* that specifies immediate rewards for performing particular actions in particular states. For example, $R(s, a)$ is the reward agents associate with performing action a in state s. Normally, receiving rewards greater than zero is considered *positive reinforcement* because it encourages taking the same actions again. Conversely, rewards less than zero convey *negative reinforcement*, which discourages similar actions.

At each time step, the agent observes the current state $s \in S$ of the environment and selects an action $a \in A$. As a result of the agent performing action a, the environment transitions to state $s' \in S$ with probability $T(s, a, s')$. The agent then observes the associated reward $R(s, a)$ and the new state s'. The process repeats indefinitely or until a final state is reached.

The goal of Reinforcement Learning is to determine what is the *best action* to perform in each of the states in the state space. Mathematically, this means finding a mapping of states to actions, $\pi : S \rightarrow A$, where $a = \pi(s)$ denotes the action a to be selected at state s. This mapping function π is often referred to as an action selection strategy, or more formally as a *policy*. Policies can be either *deterministic*, in the sense that the same action will be selected every time the agent visits the same state, or *nondeterministic*, in the sense that different actions might be selected at different times the agent passes through the same state. An

example of the former type was given above; an example of the latter can be defined as follows: $\pi : S \times A \rightarrow [0, 1]$, where $\pi(s, a) \equiv p(a \mid s)$. Note that we can encode deterministic policies using the nondeterministic notation by setting $\pi(s, a_i) = 1$ for some $a_i \in A$ and $\pi(s, a_j) = 0$ for the remaining $a_j \in A$, $j \neq i$. We will use this notation henceforth without loss of generality.

There is still the question of what is considered a *best action*. In Reinforcement Learning good actions are actions likely to yield relatively high long-term expected reward. To distinguish from the immediate reward defined earlier, we will refer to the long-term expected reward in state s as the *value* of reaching state s. We will denote this value with $V(s)$, where $V : S \rightarrow \mathbb{R}$. If the underlying MDP has a final state then the *value function* $V(s)$ can be defined simply as the sum of expected rewards starting from state s: $V(s) = \sum_a \pi(s, a) \left[R(s, a) + \sum_{s'} T(s, a, s')V(s')\right]$. If the MDP does not explicitly define final states then we can introduce a *discount factor* to deal with the infinite recursion. The resulting equation is referred to as Bellman's equation:

$$V(s) = \sum_{a \in A} \pi(s, a) \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V(s') \right) \tag{2.6}$$

where the discount factor $\gamma \in [0, 1]$ specifies how much more we value immediate rewards over delayed rewards. Note that if $\gamma = 1$ then this formulation is the same as the one defined for MDPs with a final state. We will therefore use this formulation henceforth for the sake of preserving generality. On the other extreme, if $\gamma = 0$ then the value of reaching a state is reduced to its expected immediate reward: $V(s) = \sum_a \pi(s, a)R(s, a)$. In practice, γ is usually set to a number close to 1.

Similarly, we can define the optimal value function

$$V^*(s) = \max_{a \in A} \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V^*(s') \right)$$

We may sometimes wish to explicitly distinguish between the value achieved using one policy versus another. In this case we may superscript a value function with a specific policy (e.g., V^π) to denote which policy achieves the value in question.

Using this notion of value we define the "best" action, or more formally the *optimal action*, in some state s as the action that is expected to yield the maximum value from state s onwards.[2] If we denote this action with a^*_s then we have

$$a^*_s = \operatorname*{argmax}_{a \in A} \sum_{s' \in S} T(s, a, s')V^*(s') \tag{2.7}$$

[2]Again, whenever ambiguity does not arise, we may use *the* optimal action to refer to any of a number of equally-optimal actions.

Therefore, *solving* an MDP amounts to finding the *optimal policy* (i.e., one that selects optimal actions in every state). We will denote this policy by π^*.

Many different methods exist for solving MDPs. Most of them work using Dynamic Programming, by alternating between the following two steps:

$$1) \quad \pi(s,a) \leftarrow \delta \left(a = \underset{a' \in A}{\operatorname{argmax}} \sum_{s' \in S} T(s,a',s')V(s') \right)$$

$$2) \quad V(s) \leftarrow \sum_{a \in A} \pi(s,a) \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s') \right)$$

The first step here updates the (deterministic, in this case) policy based on the current estimate of the value function. The second updates the value function based on the current estimate of the policy.

One of the most popular methods that uses the above steps is Value Iteration [1, 9]. Value Iteration searches in the space of value functions to find V^* by combining the two steps into the following one-step iterative update:

$$V(s) \leftarrow \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s') \right) \tag{2.8}$$

Using this update rule V is guaranteed to converge to V^*. [17] When this happens, we can construct a deterministic π^* using step 1) above.

Another method for solving MDPs is Policy Iteration. Policy Iteration searches in the space of policies by performing step 1) once, then repeating step 2) until convergence, then performing step 1) again and so on. We know we have converged to the optimal policy when step 1) results in no change to the current policy estimate.

One final note worth drawing attention to before leaving the context of Reinforcement Learning is that of the tradeoff between exploration and exploitation. It is often the case that learning agents know very little about the environment *a priori* and therefore do not know which actions are best to perform. As a result, they often select actions that seem optimal given the limited knowldge they have, but in fact may be sub-optimal. This is referred to as *exploitation*. During the agent's initial learning stages, exploitation is not a good strategy because it hinders (or, in extreme cases, altogether avoids) learning. On the other hand, if agents purposely select sub-optimal actions hoping to gain more information about the environment then they are not fully utilizing the information thus gained. This is referred to as *exploration* and it is often undesired because of frequently incurring opportunity

costs[3]. The *tradeoff* between exploration and exploitation refers to the fact that the two are inversely related and it is not always obvious how to achieve balance. To avoid this difficulty people often resolve to introducing a problem-specific parameter that controls this tradeoff. Usually, the parameter favours exploration in the beginning of the learning process and later shifts more and more towards exploitation. An alternative is to optimize over the tradeoff directly, which has been done by Poupart *et al.* [14] and is discussed in Chapter 3.

2.3 Linear Constraint Satisfaction

Constraint Satisfaction is the process of finding a solution to a set of constraints. Constraints are usually specified in terms of equations with unknown variables. We refer to these unknowns as *decision variables*. The goal is therefore to find a set of assignments of values to decision variables, such that all constraints are satisfied. A proposed set of assignments in general is called a *solution* and a solution which satisfies all constraints is called a *feasible solution*. It is not unusual for problems to have more than one feasible solution. However, in the case that no feasible solution exists, the problem itself is called *infeasible*.

A special case of this class of problems is observed when all constraints are linear (i.e., every term in every constraint is either a constant or a constant multiplied by the first power of a variable). In this case, we refer to the problem as a *Linear Constraint Satisfaction* problem. Finding solutions to Linear Constraint Satisfaction problems is generally easier than their non-linear variations.

In many real-world scenarios it is desirable to not just find *any* feasible solution to a set of constraints but to find the *optimal* solution. The optimal solution is defined as a feasible solution that minimizes or maximizes some function expressed in terms of decision variables.[4] This function is called the *objective function*. The process of finding the optimal solution to a Constraint Satisfaction problem is called *Constraint Programming*. Similarly, when we deal with linear constraints and a linear objective function we are dealing with *Linear Programming* problems. In the case of linear constraints but a quadratic (and convex) objective function the problem is referred to as a *Quadratic Programming* problem.

[3]Opportunity cost is the difference in costs (or profits) between the action taken and an alternative mutually exclusive action. It is borrowed from the field of Economics and used here to refer to differences in rewards.

[4]Even though there could potentially exist multiple solutions with the same degree of optimality, applications usually do not distinguish between them. Therefore, whenever we refer to *the* optimal solution, any other equally-optimal solution is acceptable.

In our research we often formulate problems using Linear Programming nomenclature. It is therefore useful to review this nomenclature. We briefly do this here. For a more detailed description please refer to [18].

Given a $1 \times J$ objective function vector $\boldsymbol{f} = [f_1, ..., f_J]$, an $I \times J$ coefficient matrix[5] $\boldsymbol{A} = [\boldsymbol{a}_1, ..., \boldsymbol{a}_I]^\mathsf{T}$, where each $\boldsymbol{a}_i = [a_{i1}, ..., a_{iJ}]$, and an $I \times 1$ boundary vector $\boldsymbol{b} = [b_1, ..., b_I]^\mathsf{T}$, the goal of Linear Programming is to find a $J \times 1$ vector $\boldsymbol{x} = [x_1, ..., x_J]^\mathsf{T}$ that

$$\begin{aligned} &\text{minimizes} && \boldsymbol{f}\boldsymbol{x} \\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & && \boldsymbol{x} \geq 0 \end{aligned} \tag{2.9}$$

While the formulation itself looks simple, it involves a number of variables and coefficients that may be easily confused. A summary of what each vector/matrix and their contents mean appears in Table 2.1.

Table 2.1: Summary of Linear Programming Terms

Term	*Meaning*
$\boldsymbol{x}$	a solution
x_j	a decision variable
$\boldsymbol{f}$	the objective function
$\boldsymbol{A}$	the coefficients matrix
$\boldsymbol{a}_i$	a vector of coefficients
a_{ij}	a coefficient
$\boldsymbol{b}$	the boundary vector
b_i	a boundary term
$\boldsymbol{a}_i\boldsymbol{x} \leq b_i$	the i-th constraint
$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$	all constraints

The above focuses on linear objective functions, but it is not difficult to extend the same notation to quadratic functions. For example, we can use an objective function of the form $\boldsymbol{x}^\mathsf{T}\boldsymbol{F}\boldsymbol{x} + \boldsymbol{f}\boldsymbol{x}$, where the coefficients of the objective function are given by the symmetric $|\boldsymbol{x}| \times |\boldsymbol{x}|$ matrix $\boldsymbol{F}$ and the $1 \times |\boldsymbol{x}|$ vector $\boldsymbol{f}$.

Note that the formulation of (2.9) minimizes (as opposed to maximizing) the objective function. Nevertheless, we can still encode maximization problems by simply optimizing over $(-\boldsymbol{f})\boldsymbol{x}$ instead of $\boldsymbol{f}\boldsymbol{x}$. The same trick can be used if a

[5]Here we denote matrices by bold uppercase letters and vectors - by bold lowercase letters.

constraint needs to be converted from an upper bound to a lower bound. Equality constraints appear as a pair of a lower bound and an upper bound constraint:

$$\boldsymbol{a}_i\boldsymbol{x} = b_i \quad \Leftrightarrow \quad \left| \begin{array}{l} \boldsymbol{a}_i\boldsymbol{x} \leq b_i \\ \boldsymbol{a}_i\boldsymbol{x} \geq b_i \end{array} \right. \quad \Leftrightarrow \quad \left| \begin{array}{l} \boldsymbol{a}_i\boldsymbol{x} \leq b_i \\ -\boldsymbol{a}_i\boldsymbol{x} \leq -b_i \end{array} \right.$$

Sometimes it is also useful to add more variables to the formulation of problems in order to facilitate computation. (How will be explained later.) Such variables are not contained in the final solution and so are referred to as *slack* variables.

In our research we often deal with overconstrained problems of the form

$$\min_{\boldsymbol{x} \geq 0} ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}|| \tag{2.10}$$

where "overconstrained" means $I \gg J$, and $||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||$ is some norm of $(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$. The norm can be thought of as the "distance" between the two vectors $\boldsymbol{A}\boldsymbol{x}$ and $\boldsymbol{b}$. However, there are many ways to interpret "distance" between vectors. In general, the p-norm of a vector $\boldsymbol{x}$, denoted $||\boldsymbol{x}||_p$, is defined as

$$||\boldsymbol{x}||_p \equiv \left(\sum_{j=1}^{J} |x_j|^p \right)^{\frac{1}{p}} \tag{2.11}$$

We consider two norms in particular: the *maximum norm* ($p = \infty$) and the *Euclidean norm* ($p = 2$).

$$||\boldsymbol{x}||_\infty = \max_j |x_j| \tag{2.12}$$

$$||\boldsymbol{x}||_2 = \sqrt{\sum_j (x_j)^2} \tag{2.13}$$

If we take the maximum norm in equation 2.10 we obtain a linear objective function. Therefore, we can use a Linear Programming method, such as the Simplex method (see Sec. 2.3.1), to find the solution.

On the other hand, if we consider the Euclidean norm we obtain a quadratic (convex) objective function[6] and therefore a Quadratic Programming problem. To find a solution we can use the method of minimizing Euclidean distance, described in Sec. 2.3.2.

[6]The square root in (2.13) can be ignored when minimizing, thus leaving us with a quadratic function.

2.3.1 The Simplex Method

Consider a Linear Programming problem of the form (2.9). Let us try to visualize it geometrically. To keep things visually manageable, suppose the problem is two-dimensional (i.e., $j = 2$) and we have only five constraints ($i = 5$). We can view the constraints as half-planes in the Cartesian plane. Then, the intersection of all such half-planes defines the space of all feasible solutions. This space is called the *feasibility region*. Refer to Fig. 2.1.

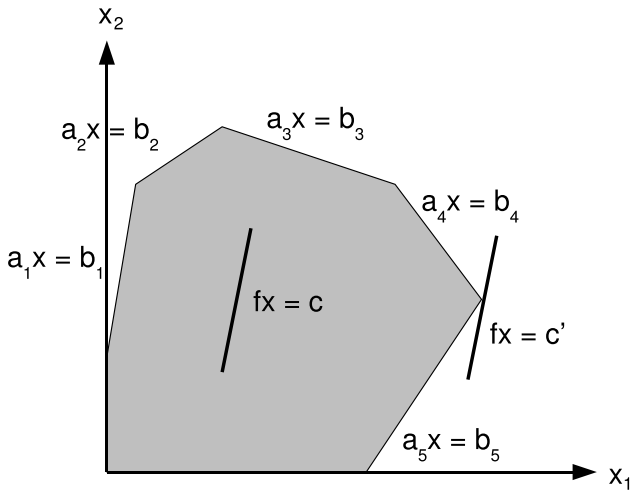

Figure 2.1: A visualization of a Linear Programming problem in 2D. The shaded (feasibility) region is bounded by the constraints $\boldsymbol{Ax} \leq \boldsymbol{b}$ and $\boldsymbol{x} \geq 0$ and contains all feasible solutions. The optimal solution is given by the point on the boundary of the feasibility region where the objective function $\boldsymbol{fx}$ achieves minimum value c'.

The Simplex method works by exploiting the fact that there always exists an optimal solution on one of the corners of the boundary of the feasibility region. It begins by selecting a corner on the boundary and evaluating the objective function at that corner. Next, it jumps to an adjacent corner, in the direction of decreasing objective function values. When it reaches a corner whose neighbours both achieve higher values, the current corner is returned as the optimal solution.

We can see (albeit not graphically) that this works in higher dimensions too. For $j \geq 3$ each constraint is represented by a half-space and the feasibility region takes the form of a convex polytope. Then, moving along the boundary is less trivial to imagine, but still as straightforward to perform computationally as in the two-dimensional case.

This is how the Simplex algorithm can be used to solve Linear Programming

problems of normal form (2.9). In addition, it is not difficult to see that the same method works for minimizing the maximum norm of the difference between two vectors (thus solving equation 2.10). By introducing a slack variable ε, we can convert (2.10) into the form of (2.9):

$$\begin{aligned} \text{minimize} \quad & \varepsilon \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} - \varepsilon\hat{\boldsymbol{e}} \leq 0 \\ & \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} - \varepsilon\hat{\boldsymbol{e}} \leq 0 \\ & \varepsilon \geq 0 \\ & \boldsymbol{x} \geq 0 \end{aligned}$$

where $\hat{\boldsymbol{e}}$ is an $I \times 1$ vector of ones. The above is clearly a Linear Programming problem and as such can be solved by the Simplex method.

For more theoretical and computational details on the Simplex method refer to [18].

2.3.2 Euclidean Distance Minimization

This method is useful when we need to solve problems of the form (2.10) when the norm is taken in Euclidean terms. If we view $f(\boldsymbol{x}) = ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2$ as the objective function in a Constraint Programming problem then the Simplex method fails because $f(\boldsymbol{x})$ is not linear in $\boldsymbol{x}$. The method described in this section overcomes this problem by exploiting the fact that $f(\boldsymbol{x})$ is quadratic:

$$\begin{aligned} f(\boldsymbol{x}) &= ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2 \\ &= (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\mathsf{T}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \\ &= \boldsymbol{x}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^\mathsf{T}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{b} + \boldsymbol{b}^\mathsf{T}\boldsymbol{b} \\ &= \boldsymbol{x}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{x}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{b} + \boldsymbol{b}^\mathsf{T}\boldsymbol{b} \end{aligned}$$

As we can see from above, $f(\boldsymbol{x})$ is quadratic and convex. Therefore, its absolute minimum can be found by setting its derivative to zero.

$$\begin{aligned} & \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = 0 \\ \Rightarrow \quad & 2\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{A}^\mathsf{T}\boldsymbol{b} = 0 \\ \Rightarrow \quad & \boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^\mathsf{T}\boldsymbol{b} \end{aligned}$$

The solution is given by solving the above system of linear equations. If the columns of $\boldsymbol{A}$ are linearly independent then its product with its transpose is invertible and

$$\boldsymbol{x} = (\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\boldsymbol{b}$$

Chapter 3

Related Work

In our research we aim to solve the Reinforcement Learning problem efficiently by incorporating a broad range of prior information. Bayesian learning provides a convenient framework to include prior information. Therefore, in this chapter, we will review previous approaches to Bayesian Reinforcement Learning, paying close attention to the type of prior information that they can take advantage of.

Bayesian Reinforcement Learning [15] is a framework that allows agents to not only act optimally but also learn the underlying model of the environment. In [15] Strens uses the idea that the uncertainty in the environment's underlying MDP model can be encoded with a probability distribution. He initially constructs a distribution over all possible MDP models, referred to as *hypotheses*, and keeps updating this distribution after observing evidence[1] at each time step. This distribution is referred to as the agent's *belief*. The policy at any time is given by sampling a hypothesis from the current belief and determining the optimal policy for that hypothesis. With time, the peak of the distribution shifts towards the true MDP and so the policy converges to the optimal one. This approach implicitly optimizes the tradeoff between exploration and expectation in the sense that agents will naturally tend to explore more often in the beginning of the learning process and less when nearing convergence.

Dearden *et al.* propose a similar framework in their work on Bayesian Q-learning [4]. They also model uncertainty the Bayesian way. However, instead of uncertainty in the environment, they model uncertainty in the *value of information* associated with performing an exploratory action. While this does not learn the underlying model of the environment, it provides a more structured approach to measuring

[1] Evidence typically consists of the immediate reward received and the next state transitioned to. In more complex scenarios it might also include observations drawn from an explicitly defined set.

the tradeoff between exploration and exploitation. Namely, the agent can directly compare the value of expected future rewards to the value of gaining exploratory information and make a choice based on this comparison. Through the course of learning the uncertainty is reduced and the selected actions converge to the optimal policy.

Another solution to Bayesian Reinforcement Learning is given by Poupart *et al.* [14], where they model the environment and the uncertainty in it with a Partially Observable MDP (POMDP). A POMDP is an extension of an MDP where the state space is not fully observable but instead can be reasoned about through an explicitly defined set of observations. In this work, the underlying MDP model of the environment is considered a part of the state space (i.e., the partially observable part) and is learned through the course of acting. Poupart *et al.* show that in this case the optimal value function is the upper envelope of a set of multivariate polynomials and develop an algorithm (BEETLE) that exploits this fact to compute an optimal policy offline. Their algorithm is practical because it allows online learning with minimal computation overhead (only belief monitoring is performed) and at the same time maximizes the expected total reward. The approach also optimizes exploration and exploitation directly, since the action selection process implicitly takes into account how the belief will change as a result of the selected action.

In a recent paper ([5]), Doshi, Roy and Pineau propose a Reinforcement Learning framework where a POMDP-modelled environment and the uncertainty in it are themselves modelled by a larger "model-uncertainty" POMDP. In their case the BEETLE algorithm does not scale because of the continuous and partially observable nature of the underlying environment, so they develop their own active learning approach. In their approach the agent selects actions that minimize the Bayes risk or, in the case when this risk is above a certain threshold, the agent admits there is too much uncertainty associated with *all* actions and asks a query to reduce this uncertainty instead of choosing a potentially disastrous action. The obvious disadvantage of this approach is the assumption that an oracle (e.g., an expert with perfect knowledge about the policy) is available online 100% of the time. It is nevertheless a viable option to consider in scenarios where the cost of active learning is lower than the cost of making frequent exploratory mistakes.

Note that the above approaches generally aim to learn about the transition parameters of the MDP model to infer a policy. Inverse Reinforcement Learning [12] takes a different approach and tries to infer the environment's reward function by observing a rational agent's behaviour. It does so by first characterizing a set of all feasible reward functions given the observed policy. This set might be too big

to work with – Ng and Russel refer to this phenomenon as *degeneracy*. They deal with degeneracy by using heuristics to select a reward function that maximally differentiates the observed policy from other suboptimal policies. The resulting problem is solved using Linear Programming. Here the notions of exploration and exploitation are irrelevant, since there is no action selection process.

A similar approach is adopted by Chajewska, Koller and Ormoneit in [2], where they learn an agent's utility function[2] from past behaviour for the purpose of predicting future behaviour. The main difference with Ng's method above is that here Bayes rule is used to update a prior over utility functions given the agent's newly observed actions as evidence. Samples from the posterior are viewed as likely utility functions.

So far we have seen how to take into account prior information over transition dynamics, reward functions *or* policies to learn about each by assuming (most of) the rest are known. Our research aims to construct a framework that takes into account priors over all three, to learn about all three.

[2]Utility function here is the same as a reward function, except that it is considered a property of the agent, not the environment.

Chapter 4

The Global Reinforcement Learning Framework

In this chapter we explain how to construct a framework that empowers the learning agent to achieve the following goals:

1. To incorporate broad prior knowledge about the learning problem so that costly actions are avoided and rate of learning is increased
2. To learn the parameters of the environment so that the agent knows how the environment will react to an action
3. To learn to act optimally so that high rewards can be achieved

How would we know if we have achieved those goals? We adopt an approach similar to Bayesian Reinforcement Learning, where the agent forms a belief about the world and updates this belief through interacting with the environment. First, we attempt to incorporate prior knowledge about both the underlying MDP model of the environment and the optimal policy. Forming a belief that reflects such prior knowledge satisfies goal 1. Next, we let the agent observe the effects of his actions on the environment, while updating his belief after each observation. If the agent's belief (or more specifically, the part of it that monitors the environment) converges to the true MDP model then goal 2 is satisfied. Finally, we tell the agent to select actions that are consistent with his current belief. Assuming goal 2 is eventually satisfied, goal 3 will be as well.

Since a learning framework is practically a program to be loaded onto an agent, it is useful to see how everything will look like from an agent's point of view. Consider the *lifecycle* of a Bayesian learning agent, illustrated in Fig. 4.1.

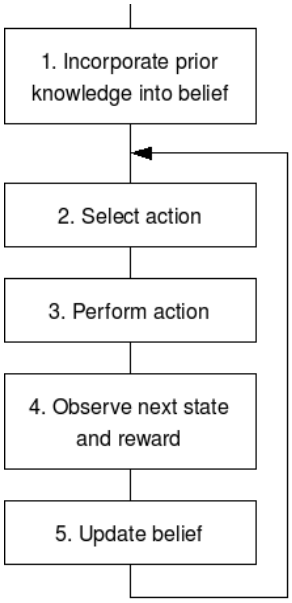

Figure 4.1: Agent's lifecycle

In the first step above, information is gathered from domain experts and used to construct the agent's initial belief. This process can be divided into two parts: *prior knowledge elicitation* and *prior knowledge incorporation*. The elicitation step involves querying experts to extract knowledge in a format that is both easily specifiable by humans and readily interpretable by machines. This step is discussed in Sec. 4.2. The incorporation step involves the mathematical work behind converting the elicited knowledge into an initial belief. This is discussed in Sec. 4.3.1. Step 2 in the agent's lifecycle is the *action selection* step and involves rules by which the current belief is used to decide which action seems optimal under the current conditions. These rules are given in Sec. 4.3.2. Steps 3 and 4 represent interactions between the environment and the agent. When learning in simulation these steps are typically simulated. When learning in the real world the steps are typically performed by the hardware comprising the logical entity we here refer to as an agent. Therefore, in both cases the details of steps 3 and 4 need not be specified by the learning framework. Finally, step 5 deals with how the agent modifies his belief about the world, based on the new evidence just observed. This is discussed in Sec. 4.3.3.

This chapter continues with definitions of some useful terms in the context of our framework. The rest of the chapter is organized as outlined above.

4.1 Terminology

We assume the learning problem takes place in an environment that can be modelled by a Markov Decision Process. (See Sec. 2.2 for a formal definition of MDPs.) The unknown parameters in an MDP are usually the transition dynamics and the rewards. Without loss of generality, we consider learning only the transition dynamics to keep the exposition simple. Therefore, we define

Environment $= \langle S, A, T() \rangle$
$S =$ set of states
$A =$ set of actions
$T : S \times A \times S \to [0, 1]$

We have purposely left out the reward function normally present in MDPs, as we will later define it as a property of the learning agent. While the sets of states and actions are known and observable, the transition function is only observable. That is, the agent does not know the value of $T(s, a, s')$ but can infer it by performing action a in state s and recording the frequency of transitions to state s'. We break this unknown function into single probability parameters $\theta_{sas'}$, one for each (s, a, s') triplet. Therefore, learning the transition function T is equivalent to learning the

Unknown transition parameters $= \theta_{sas'}$
$\theta_{sas'} \equiv T(s, a, s') \in [0, 1]$

that together form the

Transition model $= \boldsymbol{\theta}$

For every environment there is also an optimal policy (i.e., one that in any state, enables agents to gain no less reward than using any other policy).

Optimal policy $= \boldsymbol{\pi}^*$
$\boldsymbol{\pi}^* : S \times A \to [0, 1]$

The optimal policy is also unknown so we parameterize it using

Unknown policy parameters $= \pi_{sa}$
$\pi_{sa} \equiv P(a \text{ is optimal} \mid s)$

When we talk about learning a policy it is implicit that we are seeking an optimal policy. Therefore we may omit the * and simply write $\boldsymbol{\pi}$.

Learning will be performed in the environment by an agent. This agent has some belief about the world, that dictates what he thinks will happen if he performs a particular action and also, what he thinks is the best action to perform. He also has a utility function that describes how much he favours some scenarios over others. Note that such a utility function has the same purpose as a reward function in an MDP. However, this definition allows different agents to have different utilities, which we find reasonable.

Agent $= \langle B, U() \rangle$
$B =$ agent's belief
$U : S \times A \to [0, 1]$

We have considered several implementations of the belief B. One of them we formally describe in Sec. 4.3. Other alternatives we outline in Sec. 4.4.

Another issue we deal with is incorporating prior knowledge. Since we want to learn about both the transition model $\boldsymbol{\theta}$ and the optimal policy $\boldsymbol{\pi}$ it is worthwhile to incorporate prior knowledge about both entities. We assume such knowledge is available individually for each entity in terms of a probability distribution over all possibilities.

Prior knowledge over transition dynamics $= \bar{P}(\boldsymbol{\theta})$

Prior knowledge over policies $= \bar{P}(\boldsymbol{\pi})$

We use the bar in $\bar{P}(x)$ to distinguish the prior probability of x from the actual probability $P(x)$. In Sec. 4.2 we discuss how we can convert prior knowledge given by humans (domain experts) to this machine-readable form.

The goal is then to use the given $\bar{P}(\boldsymbol{\theta})$ and $\bar{P}(\boldsymbol{\pi})$ to construct a belief B that the agent will use to select actions and will update after observing each action's effects, for the purpose of eventually converging to the true $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$, while at the same time collecting optimal (or near optimal) utilities along the way.

4.2 Prior Knowledge Elicitation

Our framework incorporates prior knowledge of the form $\bar{P}(\boldsymbol{\theta})$ and $\bar{P}(\boldsymbol{\pi})$. However, note that these are second-order probability distributions (i.e., probability

distributions over probability distributions) and as such may not be very easy to specify directly by domain experts. This difficulty may be overcome by adopting a Dirichlet representation for the probability distributions and specifying their means and precisions (see Sec. 2.1.1). Therefore, in the Reinforcement Learning setting, prior knowledge may be specified using statements like:

> "With X amount of certainty, I believe that performing action a in state s has Y_i probability to lead to state s_i."

Here X can be interpreted as the number of trials that this belief is based on, and so it specifies the precision of the Dirichlet distribution for $\boldsymbol{\theta}_{sa}$. On the other hand, the Y_i values directly specify the mean.

This allows us to elicit the prior $\bar{P}(\boldsymbol{\theta})$ in a more user-friendly manner. The same technique can be used to construct $\bar{P}(\boldsymbol{\pi})$ as well.

Note, however, that the two priors $\bar{P}(\boldsymbol{\theta})$ and $\bar{P}(\boldsymbol{\pi})$ only encode information about the transition dynamics and the policy separately. But there is another piece of information that should be taken into account - the degree to which a policy is optimal for a given transition model. This third piece of prior knowledge is always implicit in every learning problem and it needs no explicit elicitation. In fact, we can encode it using Bellman's equation and the notion of value functions (2.6). Consider the following *Loss function*

$$L(\boldsymbol{\theta}, \boldsymbol{\pi}) = \max_s V^*(\boldsymbol{\theta}, s) - V^{\boldsymbol{\pi}}(\boldsymbol{\theta}, s) \tag{4.1}$$

where the optimal value function and the policy-specific value function follow the definitions given in Sec. 2.2. Intuitively, the Loss function tells us how much utility is lost by using a non-optimal policy in the worst case. In other words, given a transition model, it computes a policy's opportunity cost. We can see that for $(\boldsymbol{\theta}, \boldsymbol{\pi})$-pairs where $\boldsymbol{\pi}$ is optimal $L(\boldsymbol{\theta}, \boldsymbol{\pi})$ will be zero. For non-optimal $\boldsymbol{\pi}$ the Loss value will be greater if the actions chosen by $\boldsymbol{\pi}$ achieve lower utilities than $\boldsymbol{\pi}^*$.

It may be worth noting that there are other ways of representing the Loss function. For example, in learning problems that have a predefined start state we can drop the *max* over states and simply compute the opportunity cost from the start state on. In other scenarios, there might be states which are costly but rarely reachable. This might result in some policies having an "unfairly" high Loss value under the current "worst-case" definition. Such unfairness could be corrected by taking an expectation over states, so that the opportunity costs associated with improbable states do not have as high an impact on the Loss value. In our research,

we choose to work with the worst-case Loss to obtain an upper bound on policy suboptimality.

To summarize, our framework uses three pieces of information as prior knowledge: a prior over transition models $\bar{P}(\boldsymbol{\theta})$, a prior over policies $\bar{P}(\boldsymbol{\pi})$ and the implicit degree of optimality of each $(\boldsymbol{\theta}, \boldsymbol{\pi})$-pair. While the first two are specified by domain experts, the last is computed automatically using $L(\boldsymbol{\theta}, \boldsymbol{\pi})$. How to combine these three pieces into a whole is explained in the next section.

4.3 Agent's Belief As a Joint Mixture of Dirichlets

The agent's belief B is normally a distribution over all possibilities of the unknown parameters we wish to learn. In our case, since we are learning about both $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$, we have

$$B : \boldsymbol{\theta} \times \boldsymbol{\pi} \to [0, 1]$$

We choose to represent B with a mixture of Dirichlet distributions for several reasons. First, Dirichlets are conjugate priors - this means they are closed under Bayes' rule and will be easy to update. Second, they can be easily specified using mean and precision. Third, mixtures of Dirichlets are multi-modal (as opposed to the unimodal Dirichlet), so they can encode a broad variety of prior information. Fourth, a mixture of Dirichlets can encode any positive polynomial and is therefore very expressive.

We define B using the Dirichlet mixture below. Since B is a joint distribution, the formulation differs slightly from that of (2.5).

$$B(\boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{ij} c_{ij} \prod_{sas'} (\theta_{sas'})^{n_{isas'}} \prod_{sa} (\pi_{sa})^{m_{jsa}} \tag{4.2}$$

In this equation each of the two products can be viewed as an unnormalized Dirichlet, corresponding to an unknown entity. Each unnormalized Dirichlet is defined by its own set of hyperparameters ($\boldsymbol{n}_i$ for the transition dynamics, $\boldsymbol{m}_j$ for the policy). These products are then weighted using the weights c_{ij} and summed. The weights subsume the normalization factors of the Dirichlets, which avoids having to deal with gamma functions altogether. Furthermore, c_{ij} values are used to encode the correlations between the two unnormalized Dirichlets: for example, high weights relate distributions that are consistent.

The free parameters of this *joint mixture of Dirichlets* are therefore the hyperparameters $\boldsymbol{n}_i$ and $\boldsymbol{m}_j$ and the weights c_{ij}. We will try to initialize these parameters

to reflect our prior knowledge, use them to determine optimal actions and update them according to observed evidence.

4.3.1 Prior Knowledge Incorporation

The problem of incorporating prior knowledge amounts to constructing the agent's belief from the given prior knowledge.

Input: $\bar{P}(\boldsymbol{\theta})$, $\bar{P}(\boldsymbol{\pi})$, $L(\boldsymbol{\theta}, \boldsymbol{\pi})$

Output: $B(\boldsymbol{\theta}, \boldsymbol{\pi})$

In fact, we can view prior knowledge incorporation as a Constraint Satisfaction problem, where a set of constraints is used to enforce the correspondence of the belief to each of the three pieces of input. For the first one—the prior over transition dynamics—we can equate $\bar{P}(\boldsymbol{\theta})$ to B's marginal[1] of $\boldsymbol{\theta}$. Similarly, for the second one we can equate $\bar{P}(\boldsymbol{\pi})$ to the marginal of $\boldsymbol{\pi}$. The third one is somewhat different, in the sense that it does not specify explicit belief values. Instead, it imposes an ordering over $(\boldsymbol{\theta}, \boldsymbol{\pi})$-pairs: a pair whose Loss value is greater than another's should have a lower belief value. We can describe these rules mathematically using the following constraints.

$$\left|\begin{array}{l} \forall \boldsymbol{\theta} : \int_{\boldsymbol{\pi}} B(\boldsymbol{\theta}, \boldsymbol{\pi}) d\boldsymbol{\pi} = \bar{P}(\boldsymbol{\theta}) \\ \forall \boldsymbol{\pi} : \int_{\boldsymbol{\theta}} B(\boldsymbol{\theta}, \boldsymbol{\pi}) d\boldsymbol{\theta} = \bar{P}(\boldsymbol{\pi}) \\ \forall \boldsymbol{\theta}, \boldsymbol{\pi}, \boldsymbol{\pi'} : L(\boldsymbol{\theta}, \boldsymbol{\pi}) \geq L(\boldsymbol{\theta}, \boldsymbol{\pi'}) \Rightarrow B(\boldsymbol{\theta}, \boldsymbol{\pi}) \leq B(\boldsymbol{\theta}, \boldsymbol{\pi'}) \end{array}\right.$$

Note that in practice prior information can often be incompatible. For example, an expert might think action a_1 is the best thing to do in state s, yet at the same time believe in a transition model that achieves better utilities using a_2. We want our framework to be able to handle such cases so we "soften" the above constraints by adding a slack variable $\boldsymbol{\varepsilon}$. This variable[2] can be viewed as an error term that grows with the amount of inconsistency present in the prior knowledge. The resulting problem, therefore, is one of Constraint Programming, where we seek

[1] Since B is a joint probability distribution over both $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$, we can integrate over one of the variables to obtain a marginal distribution of the other. For example, the marginal of $\boldsymbol{\theta}$ is $\int_{\boldsymbol{\pi}} B(\boldsymbol{\theta}, \boldsymbol{\pi}) d\boldsymbol{\pi}$.

[2] Technically, in the following formulation $\boldsymbol{\varepsilon}$ is an infinite vector of variables (one for each $\boldsymbol{\theta}$, each $\boldsymbol{\pi}$ and each pair) so our notation is slightly abusive.

to minimize some norm of this error term.

$$\begin{aligned}
\text{minimize} \quad & ||\boldsymbol{\varepsilon}|| \\
\text{subject to} \quad & \forall \boldsymbol{\theta} : -\boldsymbol{\varepsilon} \leq \int_{\boldsymbol{\pi}} B(\boldsymbol{\theta}, \boldsymbol{\pi}) d\boldsymbol{\pi} - \bar{P}(\boldsymbol{\theta}) \leq \boldsymbol{\varepsilon} \\
& \forall \boldsymbol{\pi} : -\boldsymbol{\varepsilon} \leq \int_{\boldsymbol{\theta}} B(\boldsymbol{\theta}, \boldsymbol{\pi}) d\boldsymbol{\theta} - \bar{P}(\boldsymbol{\pi}) \leq \boldsymbol{\varepsilon} \\
& \forall \boldsymbol{\theta}, \boldsymbol{\pi}, \boldsymbol{\pi'} : L(\boldsymbol{\theta}, \boldsymbol{\pi}) \geq L(\boldsymbol{\theta}, \boldsymbol{\pi'}) \Rightarrow B(\boldsymbol{\theta}, \boldsymbol{\pi}) - B(\boldsymbol{\theta}, \boldsymbol{\pi'}) \leq \boldsymbol{\varepsilon}
\end{aligned}$$

Expanding the formula for B (eq. 4.2) we obtain

$$\begin{aligned}
\text{minimize} \quad & ||\boldsymbol{\varepsilon}|| \\
\text{subject to} \quad & \forall \boldsymbol{\theta} : -\boldsymbol{\varepsilon} \leq \int_{\boldsymbol{\pi}} \sum_{ij} c_{ij} \prod_{sas'} (\theta_{sas'})^{n_{isas'}} \prod_{sa} (\pi_{sa})^{m_{jsa}} d\boldsymbol{\pi} - \bar{P}(\boldsymbol{\theta}) \leq \boldsymbol{\varepsilon} \quad \text{(C1)} \\
& \forall \boldsymbol{\pi} : -\boldsymbol{\varepsilon} \leq \int_{\boldsymbol{\theta}} \sum_{ij} c_{ij} \prod_{sas'} (\theta_{sas'})^{n_{isas'}} \prod_{sa} (\pi_{sa})^{m_{jsa}} d\boldsymbol{\theta} - \bar{P}(\boldsymbol{\pi}) \leq \boldsymbol{\varepsilon} \quad \text{(C2)} \\
& \forall \boldsymbol{\theta}, \boldsymbol{\pi}, \boldsymbol{\pi'} : L(\boldsymbol{\theta}, \boldsymbol{\pi}) \geq L(\boldsymbol{\theta}, \boldsymbol{\pi'}) \Rightarrow \\
& \quad \sum_{ij} c_{ij} \prod_{sas'} (\theta_{sas'})^{n_{isas'}} \left[\prod_{sa} (\pi_{sa})^{m_{jsa}} - \prod_{sa} (\pi'_{sa})^{m_{jsa}} \right] \leq \boldsymbol{\varepsilon} \quad \text{(C3)}
\end{aligned}$$

Solving the above constraint programming problem faces two challenges. The first is performing the integration in each of the first two constraints. Fortunately, we can avoid this by sampling some $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$, so that we treat the integrals as sums over the samples. The second challenge comes from the fact that our constraints are not linear. In fact, we have variables as exponents *and* as multiplicative factors. It is not clear how such a system of constraints can be easily satisfied. Since Dirichlet mixtures are polynomials, we proceed in two steps: first, we construct basis functions (monomials) that correspond to products of unnormalized Dirichlets; second, we weigh those basis functions to satisfy the constraints.

Constructing Basis Functions

We define a basis function as a product of two unnormalized Dirichlets: one over transition dynamics and one over policies. We denote basis functions with $\beta : \boldsymbol{\theta} \times \boldsymbol{\pi} \rightarrow \mathbb{R}^+$. We aim to construct multiple basis functions, one for each weight c_{ij}. Then, the ij-th basis function is given by

$$\beta_{ij}(\boldsymbol{\theta}, \boldsymbol{\pi}) = \prod_{sas'} (\theta_{sas'})^{n_{isas'}} \prod_{sa} (\pi_{sa})^{m_{jsa}} \tag{4.3}$$

We will use the same letter to denote the individual components of basis functions.

$$\beta_i(\boldsymbol{\theta}) = \prod_{sas'} (\theta_{sas'})^{n_{isas'}}$$
$$\beta_j(\boldsymbol{\pi}) = \prod_{sa} (\pi_{sa})^{m_{jsa}}$$

Let us first look at constructing $\beta_i(\boldsymbol{\theta})$. We need to find exponents $\boldsymbol{n}_i$, such that

$$\forall \boldsymbol{\theta} : \int_{\boldsymbol{\pi}} \sum_{ij} c_{ij} \beta_i(\boldsymbol{\theta}) \beta_j(\boldsymbol{\pi}) d\boldsymbol{\pi} \approx \bar{P}(\boldsymbol{\theta})$$

Or equivalently, minimize some error terms $\varepsilon_{\boldsymbol{\theta}}$, such that

$$\forall \boldsymbol{\theta} : \frac{1}{\varepsilon_{\boldsymbol{\theta}}} \leq \frac{1}{\bar{P}(\boldsymbol{\theta})} \left[\int_{\boldsymbol{\pi}} \sum_{ij} c_{ij} \beta_i(\boldsymbol{\theta}) \beta_j(\boldsymbol{\pi}) d\boldsymbol{\pi} \right] \leq \varepsilon_{\boldsymbol{\theta}}$$

We proceed incrementally, starting with the case when $i = j = 1$. Then, there is only one basis function and the above equation reduces to

$$\forall \boldsymbol{\theta} : \frac{1}{\varepsilon_{\boldsymbol{\theta}}} \leq \frac{1}{\bar{P}(\boldsymbol{\theta})} \left[\int_{\boldsymbol{\pi}} c_{11} \prod_{sas'} (\theta_{sas'})^{n_{1sas'}} \prod_{sa} (\pi_{sa})^{m_{1sa}} d\boldsymbol{\pi} \right] \leq \varepsilon_{\boldsymbol{\theta}}$$

$$\forall \boldsymbol{\theta} : \frac{1}{\varepsilon_{\boldsymbol{\theta}}} \leq \frac{1}{\bar{P}(\boldsymbol{\theta})} \left[c_{11} \prod_{sas'} (\theta_{sas'})^{n_{1sas'}} \int_{\boldsymbol{\pi}} \prod_{sa} (\pi_{sa})^{m_{1sa}} d\boldsymbol{\pi} \right] \leq \varepsilon_{\boldsymbol{\theta}}$$

Since the product over policies is an unnormalized Dirichlet, we know its integral is equal to the multinomial Beta function (eq. 2.2). This is just another constant, so we multiply it with c_{11} and denote the product with z_1.

$$\forall \boldsymbol{\theta} : \frac{1}{\varepsilon_{\boldsymbol{\theta}}} \leq \frac{1}{\bar{P}(\boldsymbol{\theta})} \left[z_1 \prod_{sas'} (\theta_{sas'})^{n_{1sas'}} \right] \leq \varepsilon_{\boldsymbol{\theta}}$$

Now we can take the log of both sides.

$$\forall \boldsymbol{\theta} : -\log \varepsilon_{\boldsymbol{\theta}} \leq \log \left[z_1 \prod_{sas'} (\theta_{sas'})^{n_{1sas'}} \right] - \log \bar{P}(\boldsymbol{\theta}) \leq \log \varepsilon_{\boldsymbol{\theta}}$$

$$\forall \boldsymbol{\theta} : -\log \varepsilon_{\boldsymbol{\theta}} \leq \log z_1 + \sum_{sas'} n_{1sas'} \log \theta_{sas'} - \log \bar{P}(\boldsymbol{\theta}) \leq \log \varepsilon_{\boldsymbol{\theta}}$$

It does not appear possible to easily make sure the above holds true *for all* $\boldsymbol{\theta}$. However, we can sample some $\boldsymbol{\theta}$ uniformly at random and make sure it holds for those samples. Let $\boldsymbol{\theta}^k$ denote the k-th of K samples. Then, we have a set of K *linear* constraints to satisfy, while minimizing some norm of $\log \boldsymbol{\varepsilon} = \bigcup_{\boldsymbol{\theta}} \log \varepsilon_{\boldsymbol{\theta}}$. We

can see that the expression between the two inequality signs above is equivalent to the difference between vectors $\boldsymbol{Ax}$ and $\boldsymbol{b}$, where

$$\boldsymbol{x} = \begin{pmatrix} \log z_1 & n_{1s_1a_1s_1} & n_{1s_1a_1s_2} & \cdots & n_{1s_{|S|}a_{|A|}s_{|S|}} \end{pmatrix}^{\mathsf{T}}$$

$$\boldsymbol{b} = \begin{pmatrix} \log \bar{P}(\boldsymbol{\theta}^1) & \log \bar{P}(\boldsymbol{\theta}^2) & \cdots & \log \bar{P}(\boldsymbol{\theta}^K) \end{pmatrix}^{\mathsf{T}}$$

$$\boldsymbol{A} = \begin{pmatrix} 1 & \log(\theta^1_{s_1a_1s_1}) & \log(\theta^1_{s_1a_1s_2}) & \cdots & \log(\theta^1_{s_{|S|}a_{|A|}s_{|S|}}) \\ 1 & \log(\theta^2_{s_1a_1s_1}) & \log(\theta^2_{s_1a_1s_2}) & \cdots & \log(\theta^2_{s_{|S|}a_{|A|}s_{|S|}}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \log(\theta^K_{s_1a_1s_1}) & \log(\theta^K_{s_1a_1s_2}) & \cdots & \log(\theta^K_{s_{|S|}a_{|A|}s_{|S|}}) \end{pmatrix}$$

To find the $\boldsymbol{x}$ that minimizes the Euclidean norm of $\log(\boldsymbol{\varepsilon})$ then, we use the method of minimizing Euclidean distance (see Sec. 2.3.2). Thus, we can extract the exponents (hyperparameters) $\boldsymbol{n}_1$ directly from $\boldsymbol{x}$.

Now we look at the difference between $\boldsymbol{Ax}$ and $\boldsymbol{b}$. This difference is known as the *residual*. (In fact, this is the same quantity our error terms $\varepsilon_{\boldsymbol{\theta}}$ were seeking to minimize.) Let us denote the positively shifted residual (formally defined below) with $\boldsymbol{\psi}^1$. We will attempt to fit another monomial to approximate $\boldsymbol{\psi}^1$. To do this we use the same matrix $\boldsymbol{A}$ as above, but we set $\boldsymbol{b} = \log \boldsymbol{\psi}^1$. The minimization is performed in the same way as above to obtain $\boldsymbol{n}_2$.

In general, to find hyperparameters $\boldsymbol{n}_i$ we need to determine the value of each ψ^i_k,

$$\psi^i_k = \bar{P}(\boldsymbol{\theta}^k) - \sum_{j=1}^{i-1} z_j \prod_{sas'} (\theta_{sas'})^{n_{jsas'}} - \text{shift}_{i-1},$$

$$\text{where shift}_{i-1} = \min_{1 \le k \le K} \left\{ \bar{P}(\boldsymbol{\theta}^k) - \sum_{j=1}^{i-1} z_j \prod_{sas'} (\theta_{sas'})^{n_{jsas'}} \right\} - 1$$

and minimize the Euclidean distance between $\boldsymbol{Ax}$ and the new $\boldsymbol{b}$,

$$\boldsymbol{b} = \begin{pmatrix} \log \psi^i_1 & \log \psi^i_2 & \cdots & \log \psi^i_K \end{pmatrix}^{\mathsf{T}}$$

Note that taking the log of ψ^k_i is always allowed, since the shift term ensures $\boldsymbol{\psi}^i$ is strictly positive. In particular, the -1 term in the shift ensures $\boldsymbol{\psi}^i$ is never zero, while the rest ensures it is never negative. The iteration continutes until the residual drops below a certain threshold, meaning our desired level of approximation has been achieved.

Applying the same methodology to constraint (C2) we can find the hyperparameters $\boldsymbol{m}_j$. We will then have all we need to construct the basis functions $\beta_{ij}(\boldsymbol{\theta}, \boldsymbol{\pi})$. Next, we must weigh them properly so as to satisfy all three constraints.

Weighing Basis Functions

Weighing basis functions means finding coefficients c_{ij} that satisfy constraints (C1), (C2) and (C3), while minimizing $||\boldsymbol{\varepsilon}||$. Taking the infinity norm and sampling K transition models and K' policies, we have

$$\begin{aligned}
\text{minimize} \quad & \varepsilon \\
\text{subject to} \quad & \forall \boldsymbol{\theta}^k : -\varepsilon \leq \int_{\boldsymbol{\pi}} \sum_{ij} c_{ij} \beta_{ij}(\boldsymbol{\theta}^k, \boldsymbol{\pi}) d\boldsymbol{\pi} - \bar{P}(\boldsymbol{\theta}^k) \leq \varepsilon \qquad \text{(C1)} \\
& \forall \boldsymbol{\pi}^{k'} : -\varepsilon \leq \int_{\boldsymbol{\theta}} \sum_{ij} c_{ij} \beta_{ij}(\boldsymbol{\theta}, \boldsymbol{\pi}^{k'}) d\boldsymbol{\theta} - \bar{P}(\boldsymbol{\pi}^{k'}) \leq \varepsilon \qquad \text{(C2)} \\
& \forall \boldsymbol{\theta}^k, \boldsymbol{\pi}^{k'}, \boldsymbol{\pi}^{k''} : L(\boldsymbol{\theta}^k, \boldsymbol{\pi}^{k'}) \geq L(\boldsymbol{\theta}^k, \boldsymbol{\pi}^{k''}) \Rightarrow \\
& \quad \sum_{ij} c_{ij} \beta_i(\boldsymbol{\theta}^k) \left[\beta_j(\boldsymbol{\pi}^{k'}) - \beta_j(\boldsymbol{\pi}^{k''}) \right] \leq \varepsilon \qquad \text{(C3)}
\end{aligned}$$

Note that the only unknowns in the above formulation are the weights c_{ij} and the error term ε. Moreover, the objective function and all constraints are linear in terms of those unknowns. Therefore, this is a Linear Programming problem - one we can solve by the Simplex method (see Sec. 2.3.1).

Now that we have the hyperparameters $\boldsymbol{n}_i$ and $\boldsymbol{m}_j$ and the weights c_{ij}, we have constructed the Dirichlet mixture that encodes all our prior knowledge and we are ready to start using it as the agent's belief.

4.3.2 Action Selection

Action selection is the process by which an agent decides what to do in any given situation. For example, if the agent finds himself in state s, should he perform action a_1 or a_2? Should he choose the action he believes will yield the highest utility or the one that will give him a chance to learn more about the environment so that he can perhaps achieve higher utilities in the long run?

Our framework optimizes this choice directly. The agent's belief is a joint distribution over both transition models and optimal policies. Therefore, the agent can simply take the marginal over policies, sample a policy from it and choose an action as dictated by that policy.

Let us see how this is done computationally. The marginal over policies is given

by

$$\begin{aligned}\int_{\boldsymbol{\theta}} B(\boldsymbol{\theta}, \boldsymbol{\pi}) &= \int_{\boldsymbol{\theta}} \sum_{ij} c_{ij} \beta_i(\boldsymbol{\theta}) \beta_j(\boldsymbol{\pi}) d\boldsymbol{\theta} \\ &= \sum_{ij} c_{ij} \beta_j(\boldsymbol{\pi}) \int_{\boldsymbol{\theta}} \beta_i(\boldsymbol{\theta}) d\boldsymbol{\theta} \\ &= \sum_{j} \prod_{sa} (\pi_{sa})^{m_{jsa}} \sum_{i} c_{ij} \int_{\boldsymbol{\theta}} \prod_{sas'} (\theta_{sas'})^{n_{isas'}} d\boldsymbol{\theta} \\ &= \sum_{ij} z_j \prod_{sa} (\pi_{sa})^{m_{jsa}} \end{aligned} \tag{4.4}$$

where the factors z_j can be computed using the multinomial Beta function (2.4)

$$\begin{aligned} z_j &= \sum_{i} c_{ij} \int_{\boldsymbol{\theta}} \prod_{sas'} (\theta_{sas'})^{n_{isas'}} d\boldsymbol{\theta} \\ &= \sum_{i} c_{ij} \frac{\prod_{sas'} \Gamma(n_{isas'} + 1)}{\Gamma(\sum_{sas'} n_{isas'} + 1)} \end{aligned} \tag{4.5}$$

We can see the marginal has the form of a Dirichlet mixture. This makes sampling easy: we can sample a j from the distribution of z_j, then sample a policy from the j-th Dirichlet distribution and finally select an action according to the sampled policy.

This action selection method naturally progresses from high exploration to high exploitation. In the beginning the agent is uncertain about the environment and this will be reflected in his diffused belief distribution. This will result in a more random action selection process that will not necessarily yield high utilities but has a higher chance of giving the agent exploratory information. Later on, as the agent's belief converges to the true environment, the distribution will be more peaked and the selected actions will be very likely to yield maximum utilities.

4.3.3 Belief Update

After performing an action, the agent receives some evidence from the environment. Typically, such evidence includes the next state and some utility value. The agent must then update his belief to take into account this new evidence.

Formally, suppose an agent with current belief B is in state s, performs action a and receives evidence that the next state is s'. Then, the updated belief B' can

be computed as follows.

$$\begin{aligned} B'(\boldsymbol{\theta}, \boldsymbol{\pi}) &= p(\boldsymbol{\theta}, \boldsymbol{\pi} \mid s, a, s') \\ &= \frac{p(\boldsymbol{\theta}, \boldsymbol{\pi})\, p(s, a, s' \mid \boldsymbol{\theta}, \boldsymbol{\pi})}{p(s, a, s')} \\ &= z\, B(\boldsymbol{\theta}, \boldsymbol{\pi})\, \theta_{sas'} \\ &= z \sum_{ij} c_{ij} \prod_{\hat{s}\hat{a}\hat{s}'} (\theta_{\hat{s}\hat{a}\hat{s}'})^{n_{i\hat{s}\hat{a}\hat{s}'} + \delta([\hat{s},\hat{a},\hat{s}']=[s,a,s'])} \prod_{sa} (\pi_{sa})^{m_{jsa}} \end{aligned}$$

where the normalization factor z can be factored into the weights c_{ij} and computed using the multinomial Beta function for each corresponding Dirichlet.

Note that in practice the update is performed simply by incrementing the hyperparameters corresponding to the observed transition and renormalizing. Both steps can be performed in closed form, so that computation can be greatly facilitated by Dynamic Programming techniques.

The fact that the policy hyperparameters $\boldsymbol{m}_j$ are never updated may seem counterintuitive. After all, the belief update step aims to improve the agent's ability to recognize good policies, yet the above equation suggests an improvement only with respect to transition models. Let us see if this is in fact the case. Recall that the agent selects actions based on a sampled policy from the belief's marginal over policies (4.4). The question is, does updating the transition model hyperparameters impact the marginal over policies as well? From (4.5) it is evident that a change in $\boldsymbol{n}_i$ propagates through to z_j by means of the multinomial Beta function. The factor z_j itself can be thought of as a weight to the j-th policy basis function component $\beta_j(\boldsymbol{\pi})$. Updating z_j updates the probability that the sampled policy comes from the j-th component. In effect, incrementing $\boldsymbol{n}_i$ affects the marginal over policies in the following way: policy basis function components associated with transition model components that are consistent with the observed evidence become more likely. This is how the agent learns about the policy.

4.3.4 Known Limitations

Consider the belief update discussed in the previous section. It is clear that, given enough feedback from the environment, the agent's belief will converge to the underlying transition model. But is the same guaranteed to happen with the optimal policy as well?

Recall that learning about the policy is performed by increasing the weights of policy basis function components β_j that correspond to high-utility policies.

However, the components β_j are pre-computed and finite. When the agent has converged to the underlying model of the environment, he will increase the weight of the best component β_j^* and will act according to policies sampled from that component. This behaviour is not equivalent to acting according to the optimal policy $\boldsymbol{\pi}^*$ for two reasons:

1. There is no guarantee that any of the components β_j assign a high probability to $\boldsymbol{\pi}^*$. In other words, even the best β_j may favour a policy that is only similar to the optimal one.

2. Acting according to β_j^* is a nondeterministic process and even if $\boldsymbol{\pi}^*$ has a very high probability under β_j^* there is still a nonzero probability that a different policy will be sampled. This is the case even when the agent has converged to the true transition model of the environment and no further learning can be performed.

As a result, even if the agent is given infinite feedback from the environment he can only converge to the underlying transition model and not the optimal policy.

Let us think about how difficult it would be to overcome the above limitations. In order to ensure there is at least one component β_j that corresponds to the optimal policy (thus dealing with limitation 1) we would need to construct an infinite number of basis functions - one for each (possibly optimal) policy. Assuming this can be done, to ensure convergence to the optimal policy (dealing with limitation 2) we would need to force the best component β_j^* to select *only* $\boldsymbol{\pi}^*$ - that is, assign zero probability to all other $\boldsymbol{\pi}$. Since a β_j components is modelled by a Dirichlet, the only way to achieve this theoretically would be by using a precision equal to infinity.

It is clear that the above limitations cannot be easily overcome. We take a step back and consider a slightly different approach.

4.4 Agent's Belief as a Marginal Mixture of Dirichlets

The difficulties with the approach described in the previous section originate from the fact that we cannot fully encode the knowledge implicit in Bellman's equation. In fact, to do this we would need an infinite number of basis functions, so to be sure all combinations between transition models and policies are weighted properly. To

avoid these difficulties we seek a better way of encoding the knowledge implicit in Bellman's equation.

This section presents an alternative approach to modelling the agent's belief B: in terms of its transition model component and conditional policy component.

$$B(\boldsymbol{\theta}, \boldsymbol{\pi}) = B(\boldsymbol{\theta})B(\boldsymbol{\pi} \mid \boldsymbol{\theta})$$

The transition model component $B(\boldsymbol{\theta})$ denotes the probability that $\boldsymbol{\theta}$ is the underlying model of the environment. The conditional policy component $B(\boldsymbol{\pi} \mid \boldsymbol{\theta})$ denotes the probability that $\boldsymbol{\pi}$ is an optimal policy for $\boldsymbol{\theta}$. As we will see later, $B(\boldsymbol{\pi} \mid \boldsymbol{\theta})$ encodes the knowldge implicit in Bellman's equation and can be constructed from that equation alone. It is therefore static: agent's interactions with the environment do not result in any new evidence about $B(\boldsymbol{\pi} \mid \boldsymbol{\theta})$ and so it never needs to be updated. On the other hand, $B(\boldsymbol{\theta})$ needs to be updated after every action performed, to take into account observed transitions. For the same reasons as with the previous approach (see Sec. 4.3), we choose a Dirichlet mixture to encode this belief.

$$B(\boldsymbol{\theta}) = \sum_i c_i \prod_{sas'} (\theta_{sas'})^{n_{isas'}} \tag{4.6}$$

Let us see how an agent works with a belief of this type. To incorporate prior knowledge, the agent must convert prior distributions of the form $\bar{P}(\boldsymbol{\theta})$ and $\bar{P}(\boldsymbol{\pi})$, as well as Bellman's equation, into a prior belief $B(\boldsymbol{\theta})$. This process is explained in the following section. Once this prior belief is constructed the agent is free to use any existing Reinforcement Learning algorthm to take care of the action selection and belief update steps. We consider the BEETLE algortihm, as it is known to perform well in practice [14].

4.4.1 Prior Knowledge Incorporation

As before, the problem of incorporating prior knowledge amounts to constructing the agent's belief from the given prior knowledge. However, this time the output consists of two probability distributions.

Input: $\bar{P}(\boldsymbol{\theta})$, $\bar{P}(\boldsymbol{\pi})$, $L(\boldsymbol{\theta}, \boldsymbol{\pi})$

Output: $B(\boldsymbol{\theta})$, $B(\boldsymbol{\pi} \mid \boldsymbol{\theta})$

In the formulation above, both $B(\boldsymbol{\theta})$ and $\bar{P}(\boldsymbol{\theta})$ are distributions over transition models. It might seem reasonable therefore to copy $\bar{P}(\boldsymbol{\theta})$ into $B(\boldsymbol{\theta})$ and thus obtain

the agent's belief trivially. Similarly, $B(\boldsymbol{\pi} \mid \boldsymbol{\theta})$ could be constructed from $L(\boldsymbol{\theta}, \boldsymbol{\pi})$ alone by performing some trivial algebraic manipulations (explained shortly). Although this should work in practice, it is not the best way to go, as it does not take into account the prior information about policies stored in $\bar{P}(\boldsymbol{\theta})$. In order to take advantage of this additional prior information we proceed by satisfying the following set of constraints.

$$\left| \begin{array}{l} \forall \boldsymbol{\theta} : B(\boldsymbol{\theta}) = \bar{P}(\boldsymbol{\theta}) \\ \forall \boldsymbol{\pi} : \int_{\boldsymbol{\theta}} P(\boldsymbol{\pi} \mid \boldsymbol{\theta}) B(\boldsymbol{\theta}) d\boldsymbol{\theta} = \bar{P}(\boldsymbol{\pi}) \end{array} \right.$$

As before, since we cannot enforce the constraints for all $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$, we sample K transition models $\boldsymbol{\theta}^k$ and K' policies $\boldsymbol{\pi}^{k'}$ uniformly at random, and attempt to satisfy the constraints for those samples. Using the same sampled points, we can also approximate the integral in the second constraint with a sum. However, we must be careful to ensure the probabilities of sampled transition models sum up to one. Therefore, we introduce auxiliary variables $p_k \in [0, 1]$ (one for each sampled $\boldsymbol{\theta}^k$) and rewrite the constraints as follows.

$$\left| \begin{array}{l} \forall \boldsymbol{\theta}^k : \sum_i c_i \prod_{sas'} (\theta^k_{sas'})^{n_{isas'}} = \bar{P}(\boldsymbol{\theta}^k) \\ \forall \boldsymbol{\pi}^{k'} : \sum_k P(\boldsymbol{\pi}^{k'} \mid \boldsymbol{\theta}^k)\, p_k = \bar{P}(\boldsymbol{\pi}^{k'}) \end{array} \right.$$
$$p_k = \frac{\sum_i c_i \prod_{sas'} (\theta^k_{sas'})^{n_{isas'}}}{\sum_{k'} p_{k'}}$$

To satisfy the above constraints, we proceed as follows. First, we construct the conditional distribution $P(\boldsymbol{\pi} \mid \boldsymbol{\theta})$ from the given Loss function values $L(\boldsymbol{\theta}, \boldsymbol{\pi})$. (This incorporates Bellman's equation into the belief.) Next, we solve for p_k using the second constraint in the formulation above. Finally, once we have all p_k values, we solve for the Dirichlet mixture parameters c_i and $\boldsymbol{n}_i$ that satisfy both constraints.

Constructing $P(\boldsymbol{\pi} \mid \boldsymbol{\theta})$

Intuitively, $P(\boldsymbol{\pi} \mid \boldsymbol{\theta})$ is the probability that the policy $\boldsymbol{\pi}$ is optimal for the transition model $\boldsymbol{\theta}$. To construct this probability distribution we use an approach similar to Boltzmann exploration [16]. Given $L(\boldsymbol{\theta}, \boldsymbol{\pi})$, we set

$$P(\boldsymbol{\pi} \mid \boldsymbol{\theta}) = \frac{e^{-L(\boldsymbol{\theta}, \boldsymbol{\pi}) t}}{\sum_{\boldsymbol{\pi}} e^{-L(\boldsymbol{\theta}, \boldsymbol{\pi}) t}} \tag{4.7}$$

This formulation ensures that the values of $P(\boldsymbol{\pi} \mid \boldsymbol{\theta})$ are between 0 and 1, sum up to 1, and are higher for policies that are closer to optimal (i.e., having Loss values close to zero). The *temperature parameter* t can be used to control how peaked

the resulting distribution will be. Specifically, lower temperature values give more peaked distributions. Ideally, the value of t should be such that the resulting distribution is neither too uniform over all policies, nor too peaked towards a single policy and thus ignoring virtually all others. We have been successful in achieving a balance with temperature values around $t = 500$.

Solving for p_k

Next, we search for p_k values that

$$\begin{aligned} &\text{minimize} \quad \varepsilon \\ &\text{such that} \quad \forall \boldsymbol{\pi}^{k'} : -\varepsilon \leq \sum_k P(\boldsymbol{\pi}^{k'} \mid \boldsymbol{\theta}^k)\, p_k - \bar{P}(\boldsymbol{\pi}^{k'}) \leq \varepsilon \\ &\qquad\qquad \sum_k p_k = 1 \end{aligned}$$

We can see that all constraints above are linear, as is the objective function. Therefore, we use the Simplex method to solve for p_k.

Solving for c_i and $\boldsymbol{n}_i$

The parameters of the Dirichlet mixture must now satisfy the following constraints.

$$\left| \begin{array}{l} \forall \boldsymbol{\theta}^k : \sum_i c_i \prod_{sas'} (\theta^k_{sas'})^{n_{isas'}} = \bar{P}(\boldsymbol{\theta}^k) \\ \forall \boldsymbol{\theta}^k : \sum_i c_i \prod_{sas'} (\theta^k_{sas'})^{n_{isas'}} = p_k \sum_{k'} p_{k'} \end{array} \right.$$

Note that this is an overconstrained problem where even the slightest inconsistency in the given prior knowledge might result in unsatisfiability. As before, we add an error term ε to soften the constraints and thus deal with potential inconsistencies. However, it is interesting to note that here we can observe a direct numerical representation of inconsistency by examining the difference between the terms $\bar{P}(\boldsymbol{\theta}^k)$ and $p_k \sum_{k'} p_{k'}$. The bigger the discrepancy between those two terms, the more evidence there is to the fact that inconsistency exists in the given prior knowledge. We can exploit this fact to create a *consistency measure* ζ, computable by taking the infinity norm of the above difference.

$$\zeta = \max_k |\bar{P}(\boldsymbol{\theta}^k) - p_k \sum_{k'} p_{k'}| \tag{4.8}$$

The consistency measure could be a useful factor for determining whether the elicited prior knowledge should be trusted or perhaps partially ignored.

We can see that, similar to the joint mixture construction approach discussed in Sec. 4.3.1, the variables in the above constraints are either exponents or weights. We therefore proceed with the same approach used for the joint mixture: first we construct basis functions by minimizing the Euclidean distance between the terms, then we weigh the basis functions using Simplex to minimize error. The details are the same as in Sec. 4.3.1, except now we are working with the following coefficient matrix $\boldsymbol{A}$ and boundary vector $\boldsymbol{b}$.

$$\boldsymbol{A} = \begin{pmatrix} 1 & \log(\theta^1_{s_1a_1s_1}) & \log(\theta^1_{s_1a_1s_2}) & \cdots & \log(\theta^1_{s_{|S|}a_{|A|}s_{|S|}}) \\ 1 & \log(\theta^2_{s_1a_1s_1}) & \log(\theta^2_{s_1a_1s_2}) & \cdots & \log(\theta^2_{s_{|S|}a_{|A|}s_{|S|}}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \log(\theta^K_{s_1a_1s_1}) & \log(\theta^K_{s_1a_1s_2}) & \cdots & \log(\theta^K_{s_{|S|}a_{|A|}s_{|S|}}) \\ 1 & \log(\theta^1_{s_1a_1s_1}) & \log(\theta^1_{s_1a_1s_2}) & \cdots & \log(\theta^1_{s_{|S|}a_{|A|}s_{|S|}}) \\ 1 & \log(\theta^2_{s_1a_1s_1}) & \log(\theta^2_{s_1a_1s_2}) & \cdots & \log(\theta^2_{s_{|S|}a_{|A|}s_{|S|}}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \log(\theta^K_{s_1a_1s_1}) & \log(\theta^K_{s_1a_1s_2}) & \cdots & \log(\theta^K_{s_{|S|}a_{|A|}s_{|S|}}) \end{pmatrix}$$

$$\boldsymbol{b} = \begin{pmatrix} \log \bar{P}(\boldsymbol{\theta}^1) & \cdots & \log \bar{P}(\boldsymbol{\theta}^K) & \log\left(p_1 \textstyle\sum_k p_k\right) & \cdots & \log\left(p_K \textstyle\sum_k p_k\right) \end{pmatrix}^\intercal$$

This approach has the advantage of building only basis functions corresponding to transition models and using these basis functions to reason about optimal policies. Since these basis functions are constantly updated through the closed-form belief update, even in the case where erroneous prior knowledge is given the framework will still be able to converge to the true model of the environment (albeit more slowly) and select optimal actions from then on.

Chapter 5

Experiments and Results

To test the validity of both approaches described in the previous chapter, we conduct several experiments on toy problems. The experiments are preliminary. The goal is to provide a proof-of-concept demonstrating that the proposed algorithms do work in practice.

Specifically, we test for four things:

1. How well does the constructed prior belief reflect prior knowledge?
2. How informative is the consistency measure (equation 4.8)?
3. Do any of the two approaches converge to optimal behaviour in practice?
4. How do the algorithms scale with the number of unknown parameters?

To answer the above questions we perform experiments on a simple two-state two-action world. Refer to Fig. 5.1.

In this world the agent starts in state $s1$ and has the choice of performing either action $a1$ or $a2$. Either action has some probability of leading the agent to state $s2$ (30% and 50% respectively). Such a transition is desired because it will give the agent a high reward (namely, 10) for performing any action in $s2$. In fact, we can easily see that the highest rewards are obtained by staying in $s2$ for as long as possible. The highest probability of achieving this is by following a policy that selects $a2$ in $s1$ and $a1$ in $s2$. This policy is optimal for this world.

The agent may not be fully aware of the transition probabilities and optimal actions. In the beginning he will only know as much as his prior knowledge. Recall that we elicit prior knowledge in terms of prior distributions $\bar{P}(\boldsymbol{\theta})$ and $\bar{P}(\boldsymbol{\pi})$. (The third piece—the loss function $L(\boldsymbol{\theta}, \boldsymbol{\pi})$—is not elicited, but rather extracted from

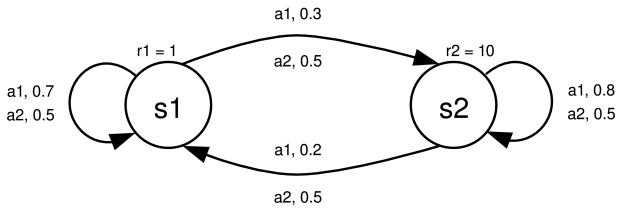

Figure 5.1: A simple test world with two states ($s1$ and $s2$), two actions ($a1$ and $a2$) and rewards ($r1$ and $r2$). Nodes in the graph represent states and edges - transitions. Transitions are stochastic: for example, performing $a1$ in $s1$ will lead back to state $s1$ with probability 0.7 and to state $s2$ with probability 0.3. Rewards are only state-dependent.

Bellman's equation.) In our experiments we test for different scenarios by giving the agent prior knowledge with various degrees of consistency. We identify three such degrees: consistent, less consistent and inconsistent. We denote the corresponding priors with $\bar{P}_C$, $\bar{P}_L$ and $\bar{P}_I$ respectively. We vary the degree of consistency by keeping the prior over transitions the same but choosing priors over policies with various degrees of optimality. For consistent prior knowledge we use Dirichlet distributions with the following parameters.

$$\bar{P}_C(\boldsymbol{\theta}) = k \prod_{sas'} (\theta_{sas'})^{n_{sas'}}$$

$$n_{111} = 14$$
$$n_{112} = 6$$
$$n_{121} = 1$$
$$n_{122} = 1$$
$$n_{211} = 2$$
$$n_{212} = 8$$
$$n_{221} = 1$$
$$n_{222} = 1$$

$$\bar{P}_C(\boldsymbol{\pi}) = k \prod_{sa} (\pi_{sa})^{m_{sa}}$$

$$m_{11} = 1$$
$$m_{12} = 50$$
$$m_{21} = 50$$
$$m_{22} = 1$$

Intuitively, we can interpret the above parameters as occurrence counts for particular events (e.g., "I believe performing $a1$ in $s1$ has a high change of leading back to $s1$, as if I had observed this happen 14 times out of $14 + 6 = 20$ trials").

The priors with lower degrees of consistency are defined as follows.

$$\bar{P}_L(\boldsymbol{\theta}) = \bar{P}_C(\boldsymbol{\theta})$$

$$\bar{P}_L(\boldsymbol{\pi}) = k \prod_{sa} (\pi_{sa})^{m_{sa}}$$

$$m_{11} = 20$$

$$m_{12} = 1$$

$$m_{21} = 10$$

$$m_{22} = 1$$

$$\bar{P}_I(\boldsymbol{\theta}) = \bar{P}_C(\boldsymbol{\theta})$$

$$\bar{P}_I(\boldsymbol{\pi}) = k \prod_{sa} (\pi_{sa})^{m_{sa}}$$

$$m_{11} = 50$$

$$m_{12} = 1$$

$$m_{21} = 1$$

$$m_{22} = 50$$

The four questions identified in the beginning of this chapter are evaluated in the following four sections respectively.

5.1 Testing Prior Belief

This experiment aims to show how well the constructed prior belief reflects prior knowledge. To do this we compare the average Euclidean distance between the true transition model and points sampled from the constructed belief versus points sampled from the prior over models. If the belief points are closer then the belief distribution must be more peaked towards the true model.

We go through the following steps:

- Construct $B(\boldsymbol{\theta})$ from $\bar{P}_C(\boldsymbol{\theta})$ and $\bar{P}_C(\boldsymbol{\pi})$ using the marginal mixture approach described in Sec. 4.4.1.
- Sample N points $\dot{\boldsymbol{\theta}}_P$ from $\bar{P}(\boldsymbol{\theta})$ and N points $\dot{\boldsymbol{\theta}}_B$ from $B(\boldsymbol{\theta})$.

- For each sampled point $\dot{\boldsymbol{\theta}}$, compute its Euclidean distance $d_{\dot{\boldsymbol{\theta}}}$ from the true transition model $\boldsymbol{\theta}^*$.

$$d_{\dot{\boldsymbol{\theta}}} = \sqrt{\sum_{sas'} (\theta^*_{sas'} - \dot{\theta}_{sas'})^2}$$

- Compute the average distances $\hat{d}^P_{\dot{\boldsymbol{\theta}}}$ of $\dot{\boldsymbol{\theta}}_P$-samples and $\hat{d}^B_{\dot{\boldsymbol{\theta}}}$ of $\dot{\boldsymbol{\theta}}_B$-samples.

$$\hat{d}_{\dot{\boldsymbol{\theta}}} = \frac{1}{N} \sum_{\dot{\boldsymbol{\theta}}} d_{\dot{\boldsymbol{\theta}}}$$

- Compare the two averages. If $\hat{d}^B_{\dot{\boldsymbol{\theta}}}$ is smaller then the knowledge in $\bar{P}(\boldsymbol{\pi})$ has been successfully incorporated to reduce uncertainty in the agent's belief.

- Repeat from the beginning for I number of iterations, averaging out the results.

There are a number of parameters that affect the quality of the constructed prior. We use the following values in this experiment.

- $K = K' = 100$ (Number of uniformly sampled transition models and policies used for constructing the prior. See Sec. 4.4.1.)

- $N = 1000$ (Number of sampled transition models used for calculating average distance to true transition model.)

- EDM threshold $= 10^{-4}$ (Convergence threshold for Euclidean distance minimization. See Sec. 4.3.1.)

- $I = 50$ (Number of iterations the experiment is performed.)

We obtain the following results.

$$\hat{d}^P_{\dot{\boldsymbol{\theta}}} = 0.7155 \qquad \hat{d}^B_{\dot{\boldsymbol{\theta}}} = 0.5645$$

What these numbers mean is that samples from the belief are, on average, 21% closer to $\boldsymbol{\theta}^*$. This shows that constructing a belief distribution from both prior distributions $\bar{P}(\boldsymbol{\theta})$ and $\bar{P}(\boldsymbol{\pi})$ does in fact incorporate more prior information than simply using $\bar{P}(\boldsymbol{\theta})$ alone.

5.2 Testing Consistency Measure

This experiment tests the consistency measure ζ. We compute and compare the value of ζ for the three pairs of prior distributions defined in the beginning of this chapter.

We go through the following steps:

- Given $\bar{P}_C(\boldsymbol{\theta})$ and $\bar{P}_C(\boldsymbol{\pi})$, compute ζ_C as described in Sec. 4.4.1.
- Given $\bar{P}_L(\boldsymbol{\theta})$ and $\bar{P}_L(\boldsymbol{\pi})$, compute ζ_L.
- Given $\bar{P}_I(\boldsymbol{\theta})$ and $\bar{P}_I(\boldsymbol{\pi})$, compute ζ_I.
- Repeat from the beginning for I number of iterations, averaging out the results.

The parameter values we use for this experiment are as follows. We decrease the number of sampled transition models and policies to avoid issues with scalability.

- $K = K' = 10$
- EDM threshold $= 10^{-4}$
- $I = 100$

We obtain the following results.

$$\zeta_C = 3.6627 \qquad \zeta_L = 4.0416 \qquad \zeta_I = 5.248$$

Recall that the consistency measure ζ by definition measures the largest inconsistency between the prior probabilities of a sampled transition model according to the prior over transition models alone and according to the prior over policies coupled with Bellman's equation. It is no surprise then that the consistent priors achieve the lowest amount of inconsistency.

Future work could explore how to use this measure to incorporate only prior information that is consistent, by automatically removing inconsistencies. This is discussed in more detail in the last section of this document.

5.3 Testing Convergence to Optimal Behaviour

Here we check if the two approaches described in Chap. 4 eventually lead the agent to converge to optimal behaviour (i.e., start achieving maximum rewards). Consider three agents: one that acts using a joint mixture belief, one using a marginal mixture belief and one using the true optimal policy. We want to see how the rewards achieved by the three agents compare to each other. The third agent is guaranteed to achive maximum rewards (by definition of "optimal policy"); the other agents should start out by receiving lower rewards (during their exploration stages) but the key thing to note is whether they eventually start gaining rewards on par with the third agent. The experiment proceeds as follows.

Agent 1: Using a joint belief.

- Construct $B(\boldsymbol{\theta}, \boldsymbol{\pi})$ from $\bar{P}_L(\boldsymbol{\theta})$ and $\bar{P}_L(\boldsymbol{\pi})$ using the joint mixture approach described in Sec. 4.3.1.
- Act for N steps, by sampling a policy from the marginal at each step (see Sec. 4.3.2).
- Record the rewards received. (There will be N of them.)
- Repeat from the beginning for I number of iterations, averaging out the results.

Agent 2: Using a marginal mixture belief.

- Construct $B(\boldsymbol{\theta})$ from $\bar{P}_L(\boldsymbol{\theta})$ and $\bar{P}_L(\boldsymbol{\pi})$ using the marginal mixture approach described in Sec. 4.4.1.
- Act for N steps using the BEETLE algorithm [14].
- Record the rewards received.
- Repeat from the beginning for I number of iterations, averaging out the results.

Agent 3: Using the optimal policy

- Construct the optimal policy $\boldsymbol{\pi}^*$ using Value Iteration (2.8).
- Act for N steps using $\boldsymbol{\pi}^*$.

- Record the rewards received.
- Repeat from the beginning for I number of iterations, averaging out the results.

Here we use the same parameter values as in Sec. 5.1, with the following additions.

- VI threshold $= 10^{-4}$ (Convergence threshold for the Value Iteration algorithm.)
- $\gamma = 0.99$ (Discount factor. See Sec. 2.2.)
- $N = 100$ (Number of steps each agent performs.)

In the end, the output consists of three vectors: the sequences of average rewards obtained by the three agents. We plot these vectors on the same graph - see Fig. 5.2.

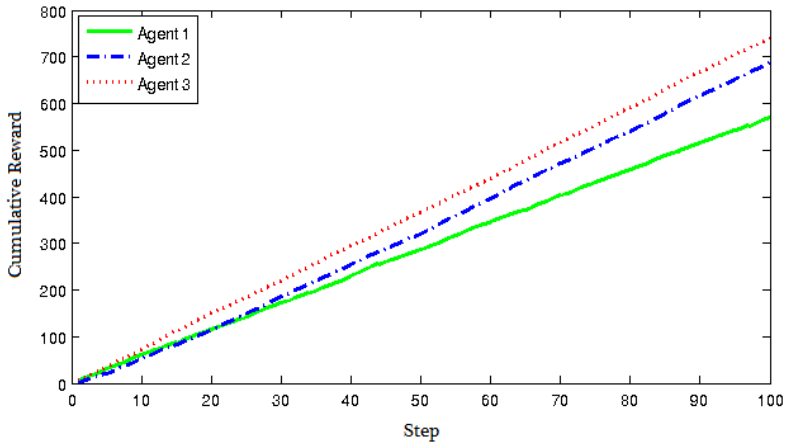

Figure 5.2: Rewards achieved by agents. Agent 1 uses a joint mixture belief, Agent 2 uses a marginal mixture belief, Agent 3 uses the optimal policy.

Agent 2, using a marginal mixture belief, converges to the optimal policy reasonably quickly. This is evident from the fact that (somewhere near the 15th step) the reward curve of Agent 2 becomes roughly parallel to the reward curve of Agent 3. This means Agent 2 has learned the optimal policy and will behave optimally from then on. On the other hand, Agent 1 who uses a joint mixture belief is not so lucky. He converges to a policy that is suboptimal and is unable to match the reward slope of the other two agents. This is due to the limitations of the joint mixture approach, discussed in Sec. 4.3.4.

5.4 Testing Scalability

Here we test how the algorithms scale with the number of unknown parameters. Since the unknown parameters are transition and action probabilities, their number depends on the size of the state and action spaces. For example, in the two-state two-action test world we defined earlier, there are $2 * 2 * 2$ unknown transition parameters and $2 * 2$ unknown policy parameters, for a total of 12 parameters to be learned. If we increase the number of states by 1, we obtain $3 * 2 * 3 + 3 * 2 = 24$ parameters. The question is how much longer would the algorithms take to accommodate such an increase?

To test this, we go through the following steps:

- Construct $B(\boldsymbol{\theta})$ from $\bar{P}_L(\boldsymbol{\theta})$ and $\bar{P}_L(\boldsymbol{\pi})$ using the joint and marginal mixture approaches.
- Record the time taken by each approach.
- Add a copy of state $s2$ to the state space, modifying U, $\bar{P}_L(\boldsymbol{\theta})$ and $\bar{P}_L(\boldsymbol{\pi})$ accordingly.
- Repeat from the top.

Here we use the same parameter values as in Sec. 5.1.

As evident from Fig. 5.3, constructing the belief using either method takes less than 30 minutes, even for problems approaching 100 unknown parameters. This indicates that the algorithms scale reasonably well with increasing the size of our toy problem. It remains to be seen whether the same success can be achieved with real-world problems.

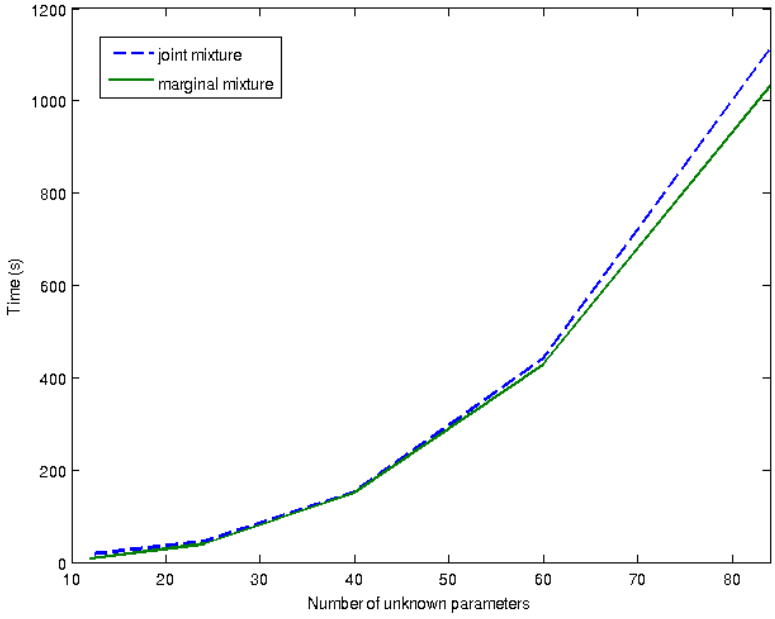

Figure 5.3: Time required for belief construction.

Chapter 6

Conclusions

This thesis described the first steps towards a reinforcement learning framework that incorporates a broad variety of prior knowledge in a principled way to help agents quickly learn to act optimally. Specifically, the framework accepts as input two prior distributions: one over transition models of the environment and one over courses of actions (policies). It then combines this knowledge with the information implicit in Bellman's equation to construct the agent's initial belief. Agents act based on the current state of this belief. The belief is continually updated using Bayes' Rule to take into account the observed evidence from the environment after each action. In this way, the agent learns the underlying transition model of the environment and converges to a specific policy. (Discussions on optimality follow shortly.)

This approach naturally progresses from high exploration to high exploitation. In the beginning the agent is uncertain about the environment and this will be reflected in his diffused belief distribution. This will result in a more random action selection process that will not necessarily yield high utilities but has a higher chance of giving the agent exploratory information. Later on, as the agent's belief converges to the true environment, the distribution will be more peaked and the selected actions will be very likely to yield maximum utilities.

We presented two alternative methods for encoding the agent's belief: as a joint mixture of Dirichlet distributions and as a marginal mixture of the same. Let us now see if the two methods satisfy the three goals we outlined in the beginning of Chapter 4:

1. To incorporate broad prior knowledge about the learning problem

    - Both methods achieve this goal by incorporating prior information over both policies and transition models.

2. To learn the parameters of the environment

    - Both methods achieve this goal by updating the hyperparameters of the belief distributions to take into account observed transitions. This is guaranteed to lead to convergence to the true model of the environment.

3. To learn to act optimally

    - The joint mixture method fails here, as it converges to a probability distribution of policies, which may at best assign a high sampling chance to the optimal policy. This is true even after the agent converges to the true model of the environment. This leads to suboptimal performance, as seen in Fig. 5.2.
    - The marginal mixture method achieves success here, as convergence to the true model guarantees convergence to the optimal policy (using the BEETLE algorithm). And convergence to the true model is guaranteed by the Bayesian belief update.

The marginal mixture approach is clearly the superior of the two and is therefore the preferred belief construction method for the Global Reinforcement Learning framework.

We also showed that incorporating broad prior knowledge is beneficial. Specifically, taking into account prior knowledge about both transition models and policies gives the agent a mode peaked belief distribution (and therefore increased rate of learning) than taking into account prior knowledge about transition models alone. This seems intuitive, but it was nevertheless worthwhile to verify experimentally (Sec. 5.1).

In the toy world used for our experiments, the Global Reinforcement Learning framework performs well. However, more work is needed to assess the framework's usefulness in real-world problems. The following section discusses possible directions for such work.

6.1 Future Work

We identify several directions for future research: (1) real world testing, (2) further broadening the scope of accepted prior knowledge, (3) generalizing applicability to partially observable domains, (4) automatically rejecting inconsistencies in prior knowledge and (5) investigating how the number of sampled points affects the precision of the belief construction process.

(1) Real world testing is an important validation aspect of any framework. As such, it seems like the next logical step for our research. Specifically, it would be interesting to see how well Global Reinforcement Learning collects rewards in an everyday scenario (e.g., the hand-washing problem [8]) and compare the results to an already established approach in that domain (e.g., BEETLE).

(2) Further broadening the scope of accepted prior knowledge can be achieved in at least two ways. One, we could extend the theory to allow incorporation of and learning about the utility (reward) function. So far we have assumed the agent has his own utility function that he is fully aware of. However, in many situations agents are uncertain about the utility function and only have some prior belief about it. The rest must be learned through interactions with the environment. This should not be too difficult to implement in Global Reinforcement Learning, since evidence about utilities is received as feedback from the environment in exactly the same way as evidence about the transition model.

Two, we could incorporate other types of prior knowledge. Currently, we have only considered eliciting prior knowledge in terms of precisions and means of Dirichlet distributions over transition models and policies. However, in many scenarios we might also wish to encode more particular knowledge, such as absolute rules (e.g., "$a1$ should never be performed"), comparison knowledge (e.g., "$a1$ is better than $a2$", "$s1$ is more likely than $s2$") and certainty intervals (e.g., "the probability that the next state will be s is anywhere between x and y"). Such prior knowledge can be easily represented by a (set of) constraint(s) and should be applicable to our belief construction process.

(3) Partially observable domains are domains in which the state space is not fully observable by agents. Instead, agents receive observations from the environment and must use them to infer the current state. Such domains are especially common for robotic agents, where the robot's position in the world must be inferred through the set of observations acquired from an onboard camera, sonar or laser range-finder. Fortunately, there already exists a framework for modelling partially observable domains. It is the Partially Observable extension to the Markov Decision Process: POMDP [10]. It should be possible to extend the Global Reinforcement Learning framework to use a POMDP model instead of MDP, thus empowering agents to incorporate broad prior knowledge in partially observable domains.

(4) There is also the question of what to do if the agent is presented (presumably by domain experts) with wrong or otherwise inconsistent prior knowledge. There are several obvious ways to deal with this, ranging from constructing a "best fit" prior belief to interactively querying domain experts for clarifications. Perhaps an even better alternative is to attempt to use the consistency measure ζ to determine

which constraints in the belief construction process are causing the inconsistencies. It is likely that the inconsistency-causing constraints are misspecified and should not be taken into account. By rejecting these constraints we should be able to construct a consistent belief from the remaining (presumably correct) constraints.

(5) The belief construction process in both the joint and marginal mixture approaches relies on sampled points to approximate integrals and satisfy universal constraints. As such, it is expected that the more points are sampled, the more accurately the constructed belief will represent prior knowledge. Future research could explore how this accuracy can be measured and determine a theoretical bound on the optimal number of sampled points necessary to achieve a specific accuracy level.

Appendix A: Symbols Glossary

a	action in a Markov Decision Process
c	weight of a Dirichlet in a Dirichlet Mixture
d	Euclidean distance between sampled points and true data
e	set of observed evidence; elementary vector
f	objective function in a Constraint Programming problem
h	hypothesis
i	counter/index
j	counter/index
k	normalization constant; counter/index
m	mean of a Dirichlet distribution
s	state in an MDP; precision of a Dirichlet distribution
x	unknown variable
y	unknown variable
z	weight coefficient
A	coefficient matrix in a Linear Programming problem; action space in a Markov Decision Process
B	the Beta function; agent's belief in Global Reinforcement Learning
I	upper limit of a counter or index; number of hyperparameters in a distribution model
J	upper limit of a counter or index
N	number of experiments, indices, variables or other quantities we wish to enumerate
R	reward function in a Markov Decision Process
$\mathbb{R}$	the set of real numbers
$\mathbb{R}^+$	the set of positive real numbers
S	state space of a Markov Decision Process
T	transition function in a Markov Decision Process
V	value function used in a Markov Decision Process

α	hyperparameter of a probability distribution model	(alpha)
β	basis function (or a component thereof)	(beta)
γ	discount factor in Bellman's equation	(gamma)
δ	the Kronecker delta function	(delta)
ε	error term; residual in the context of Euclidean distance minimization	(epsilon)
ζ	consistency measure in a marginal mixture belief	(zeta)
θ	unknown value whose probability distribution we wish to model	(theta)
κ	used as an additional temporary counter when i, j, k, a, s are already taken	(kappa)
ξ	state of information	(xi)
π	policy of an agent in an environment modelled by a Markov Decision Process	(pi)
χ	augmented set of decision variables in a Linear Programming problem	(chi)
ψ	positively shifted residual	(psi)
Γ	the Gamma function	(gamma)
Θ	unknown variable whose probability distribution we wish to model	(theta)
Σ	arithmetic summation operator	(sigma)
Π	arithmetic product operator	(pi)

References

[1] R. Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6:679–684, 1957. 12, 14

[2] U. Chajewska, D. Koller, and D. Ormoneit. Learning an agent's utility function by observing behavior. In *Proceedings of the 18th International Conference on Machine Learning*, pages 35–42. Morgan Kaufmann, San Francisco, CA, 2001. 22

[3] R.H. Crites and A.G. Barto. Improving elevator performance using reinforcement learning. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1017–1023. The MIT Press, 1996.

[4] R. Dearden, N. Friedman, and S.J. Russell. Bayesian Q-learning. In *AAAI/IAAI*, pages 761–768, 1998. 20

[5] F. Doshi, N. Roy, and J. Pineau. Reinforcement learning with limited reinforcement: Using Bayes risk for active learning in POMDPs. In *Proceedings of the 10th International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, FL, 2008. 21

[6] A. Gelman, J. Carlin, H. Stern, and D. Rubin. *Bayesian Data Analysis*. Chapman & Hall, 1995. 8

[7] D. Heckerman. A tutorial on learning with Bayesian networks. Technical report MSR-TR-95-06, Microsoft Research, Redmond, WA, 1995. Revised June 96. 6

[8] J. Hoey, P. Poupart, C. Boutilier, and A. Mihailidis. POMDP models for assistive technology. Technical report, IATSL, 2005. 52

[9] R.A. Howard. Dynamic programming and Markov processes. *The M.I.T. Press*, 1960. 14

[10] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. Technical Report CS-96-08, 1996. 52

[11] L.P. Kaelbling, M.L. Littman, and A.P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[12] A.Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, pages 663–670. Morgan Kaufmann, San Francisco, CA, 2000. 21

[13] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1025–1030, Acapulco, Mexico, 2003.

[14] P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 697–704, New York, NY, 2006. ACM. 4, 15, 21, 37, 46

[15] M.J.A. Strens. A Bayesian framework for reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, pages 943–950, San Francisco, CA, 2000. Morgan Kaufmann Publishers Inc. 20

[16] R.S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Conference on Machine Learning*, pages 216–224, 1990. 38

[17] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. 14

[18] R.J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, Hingham, MA, 1997. 16, 19

[19] M. Yamamoto and K. Sadamitsu. Dirichlet mixtures in text modeling. Technical report CS-TR-05-1, University of Tsukuba, Tsukuba, Ibaraki, Japan, 2005.