Bayesian exploration in Markov decision processes

Pablo Samuel Castro

Master of Science

School of Computer Science

McGill University

Montreal,Quebec

2007-06-18

A thesis submitted to McGill University
in partial fulfillment of the requirements of
the degree of Master of Science

DEDICATION

This thesis is dedicated to my daughter, Sofía Elena.

ACKNOWLEDGMENTS

I thank the following people from the bottom of my heart for making this thesis possible.

I would like to start off thanking my supervisor, Doina Precup, for being so supportive and encouraging throughout these two years of research. She always provided insightful feedback and advice which helped me grow as a researcher. She gave me enough freedom to explore my own ideas while making sure I was on the right track. I am grateful for all our discussions, co-writing and the editing of this thesis.

My co-supervisor, Prakash Panangaden, has been a great source of inspiration for me. His fluid and intuitive way of explaining the most complicated topics has certainly set a high standard for my pedagogical pursuits. He continuously challenged me by introducing me to new mathematical concepts and interesting problems. Thanks to him I have grown more confident as an aspiring mathematician and have developed a love for pure theory.

I would also like to thank Joelle Pineau for her help with the work presented in chapter 5. Her useful insights and encouragement were very helpful.

Many thanks are due to everybody in the School of Computer Science. In particular, the professors who taught me so much and the staff which facilitated everything for me. I would like to thank everyone in the Reasoning and Learning Lab for all the interesting discussions and chocolates! I would especially like to thank Norm for always providing guidance and encouragement, Amin for always

being willing to help, Marc for dealing with all my software requests, and Stephane for translating the abstract into French.

A huge thanks goes to all my friends in Montreal, Quito and elsewhere for all the fun times. A big thanks also goes to everyone from ecos de portoalegre, playing with them is one of the things I enjoy most!

I am very grateful for the financial support received for part of my studies. This support was provided by NSERC through a research grant of my supervisor, Doina Precup.

My family has always been an endless source of encouragement, support and inspiration. My parents Samuel and Rocío have always believed in me and supported all of my dreams. I could not imagine a better environment to grow up in, and I owe everything to them. My brother Álvaro's honest pursuit of truth and knowledge and my sister Melissa's talent and determination continue to inspire and drive me. I would also like to to thank the rest of my family for believing in me and for always being ready to help. ¡Gracias a todos y todas!

Finally, I would like to thank my wife Michelle and daughter Sofía Elena for being so patient with me while writing this thesis. Michelle's endless love, caring and sincerity replenishes my energy every day. I thank her so much for taking care of me and for fulfilling my life! Her help editing this thesis was also invaluable. Sofía Elena brings a ray of sunshine into my life. I am so grateful for her smiles and for reminding me of the beauty of life.

ABSTRACT

Markov Decision Processes are a mathematical framework widely used for stochastic optimization and control problems. Reinforcement Learning is a branch of Artificial Intelligence that deals with stochastic environments where the dynamics of the system are unknown. A major issue for learning algorithms is the need to balance the amount of exploration of new experiences with the exploitation of existing knowledge. We present three methods for dealing with this exploration-exploitation tradeoff for Markov Decision Processes. The approach taken is Bayesian, in that we use and maintain a model estimate. The existence of an optimal policy for Bayesian exploration has been shown, but its computation is infeasible. We present three approximations to the optimal policy by the use of statistical sampling.

The first approach uses a combination of Linear Programming and Q-learning. We present empirical results demonstrating its performance. The second approach is an extension of this idea, and we prove theoretical guarantees along with empirical evidence of its performance. Finally, we present an algorithm that adapts itself efficiently to the amount of time granted for computation. This idea is presented as an approximation to an infinite dimensional linear program and we guarantee convergence as well as prove strong duality.

ABRÉGÉ

Les processus de décision Markoviens sont des modèles mathématiques fréquemment utilisés pour résoudre des problèmes d'optimisation stochastique et de contrôle. L'apprentissage par renforcement est une branche de l'intelligence artificielle qui s'intéresse aux environnements stochastiques où la dynamique du système est inconnue. Un problème majeur des algorithmes d'apprentissage est de bien balancer l'exploration de l'environnement, pour acquérir de nouvelles connaissances, et l'exploitation des connaissances acquises. Nous présentons trois méthodes pour obtenir de bons compromis exploration-exploitation dans les processus de décision Markoviens. L'approche adoptée est Bayésienne, en ce sens où nous utilisons et maintenons une estimation du modèle. L'existence d'une politique optimale pour l'exploration Bayésienne a été démontrée, mais elle est impossible à calculer efficacement. Nous présentons trois approximations de la politique optimale qui utilise l'échantillonnage statistique.

La première approche utilise une combinaison de programmation linéaire et de l'algorithme "Q-Learning". Nous présentons des résultats empiriques qui démontrent sa performance. La deuxième approche est une extension de cette idée, et nous démontrons des garanties théoriques de son efficacité, confirmées par des résultats empiriques. Finalement, nous présentons un algorithme qui s'adapte efficacement au temps alloué pour le calcul de la politique. Cette idée est présentée comme une approximation d'un programme linéaire à dimension infini; nous garantissons sa convergence et démontrons une dualité forte.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

Life is the sum of all your choices.

- Albert Camus

Decision making is an essential part of a person's life and is the main determining factor for the kind of life that person will live. Decisions range from simple ones such as whether to go grocery shopping today or tomorrow, to more complex ones such as whether one should continue graduate studies or get a high paying job. The "goodness" of the choice made is proportional to the amount of experience available for the current situation. In the first example, more experience will allow one to better gauge the impact of putting off grocery shopping for another day, given the current household situation. The first couple of times one will inevitably make a couple of 'bad' choices, but given that we can learn from our mistakes, these bad choices should decrease.

In the pursuit of experience, one needs to *test the waters* in order to properly gauge the effect of the different avaliable choices. Consider a student that has just arrived in a new city and has already found a route to get from home to school in a reasonable amount of time. Every morning the student may ask herself the same question: *Is there a faster way to get to school that would allow me to sleep in more?* She must choose between two types of behavior: cautious and exploratory. A cautious student will be content with the usual route as she is

guaranteed to arrive to class on time. This type of behavior is sometimes referred to as *exploitative*, since the person is exploiting the current knowledge he or she already has. An exploratory student will try new paths in the hope of stumbling across a faster one, at the risk of getting lost or arriving late to class. Adhering constantly to one type of behavior is usually not optimal since a completely exploitative behavior may settle for something mediocre, while a completely explorative behavior may never stabilize or lead to bad results.

In this thesis we deal with the problem of balancing exploration and exploitation, specifically applied to intelligent agents. An intelligent agent is "*anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors*" (Russell and Norvig, 1995). Specifically we are dealing with agents that must choose the way they will act, and as mentioned above, we would like them to make "good" choices. This notion of preference implies there should be an ordering of the action choices based on their effects. **Utility theory** is a convenient framework for dealing with preferences by assigning utilities to the various action choices. A **utility** is a quantitative measure of the "goodness" of the various actions.

In most problems the effects of the actions are not deterministic, thus, there is some degree of uncertainty. This uncertainty can come from either incomplete knowledge of the environment or from simplifying assumptions. An example of a situation where one has incomplete knowledge of the environment is a slot machine, since perfect knowledge of its outcomes would bankrupt the casino! An example of simplifying assumptions is the roll of a fair die; one could

hypothetically predict the exact outcome by considering all the physical forces affecting the die, but rather than attempt to solve this impractical problem it can be simplified by assigning equal chance to the six possible outcomes. **Probability theory** is a mathematical framework for quantifying uncertainty, assigning a probability (a real number between 0 and 1) to each element of the set of all possible outcomes.

Decision theory is the field combining utility and probability theory. Decision theory is studied in many fields, but we shall consider it in the context of Reinforcement Learning, a branch of Artificial Intelligence that has recently received much interest. Arriving at an optimal balance between exploration and exploitation is a key problem in this field. In this thesis we will present various methods for pursuing this balance within a mathematical framework that is widely used for Reinforcement Learning.

1.1 Statement of Originality

The first algorithm presented was previously published in refereed conference proceedings (Castro and Precup, 2007). In addition to containing a more extensive literature review, we present two additional, novel algorithms. The first of these is a continuation of the algorithm of (Castro and Precup, 2007) with formal guarantees for near-optimal performance. Just as the algorithm of (Castro and Precup, 2007), it is based on the idea of sparse sampling introduced in (Kearns et al., 1999). However, it is more faithful to the original algorithm, and it is because of this that formal guarantees of near-optimality are achieved.

The last algorithm we present is a dynamic algorithm whose solution quality is directly proportional to the amount of time allowed for computation. This is an improvement over most of the existing algorithms, whose parameters must be decided beforehand. The problem is phrased as an infinite dimensional linear program and we present a sequence of approximations with guarantees of convergence. The algorithm takes advantage of this sequence and uses previous solutions in the sequence to speed up its current computation. We also prove the absence of a duality gap for the infinite dimensional linear program in question as well as for a general class of Markov decision processes. This result is significant in its own right, as the finiteness of the state space is always assumed when solving a Markov decision problem using linear programming (Puterman, 1994),(Bertsekas and Tsitsiklis, 1996).

We demonstrate the strong empirical performance of all our methods on the problems in (Castro and Precup, 2007) as well as some larger problems, thus demonstrating that our algorithms benefit from both theoretical guarantees and strong empirical performance. This is an important combination not present in many existing algorithms.

1.2 Thesis Outline

This thesis is organized as follows. In chapter 2 we present the background necessary for our decision making problem. In chapter 3 we present the formal theory characterizing the problem of balancing exploration and exploitation strategies on which our methods will be based. In chapters 4, 5 and 6 we present three different methods for producing near-optimal decisions with different formal

guarantees. We evaluate each method empirically and compare them to existing algorithms. Finally, in chapter 7 we provide some discussion of our work, along with contributions and future work.

CHAPTER 2
Background

In this chapter we present the necessary background for this thesis. We first introduce Markov Decision Processes (MDPs), which are the mathematical framework used for modeling decision making in Reinforcement Learning. We present the main algorithms for computing the value of states when the model is known, along with some approximation schemes. We then introduce Reinforcement Learning (RL), which is a field concerned with the creation of agents which *learn* how to make good decisions.

2.1 Markov Decision Processes

A Markov Decision Process (MDP) (Puterman, 1994) is a four tuple: $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}\}$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of available actions[1] , $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ denotes the transition probabilities between states, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function. If the agent is in state s and performs action a, then $\mathcal{P}(s, a, \cdot)$ is the distribution over next possible states and $\mathcal{R}(s, a)$ is the expected reward received. In this thesis we assume that $\mathcal{S}$ is finite or countable and $\mathcal{A}$ is finite.

[1] We acknowledge that many authors denote by $A_s \subset \mathcal{A}$ the set of actions available from state $s \in \mathcal{S}$. For simplicity in this thesis we assume $A_s = \mathcal{A}$ for all $s \in \mathcal{S}$. However, the results can easily be adapted to the more general case.

The following assumption is necessary for some of the theoretical results presented in this thesis.

Assumption 2.1.1. *The rewards are positive and bounded. That is, the reward function is defined as* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto [0, R_{max}]$

The boundedness assumption is standard in the MDP literature. The positivity assumption is not problematic, as any reward function can be scaled upwards in order to make it positive. This translation does not affect the ordering of the states by their optimal values, and thus, does not affect the optimal policy.

If $\mathcal{S}$ is finite, the transition probabilities between states can be stored in a tabular format, *i.e.* in an $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ array $\mathbf{P}$, whose (i, a, j)th entry is the probability of transitioning from state i to state j under action a. This representation is acceptable for state spaces which are not too big, but becomes impractical as the size of the state space increases. In section 2.4 we will consider an alternate representation for the transition probabilities.

The behavior of an agent in an MDP can be modeled using the notion of a policy. A deterministic stationary policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ is one that determines what action to take depending solely on the current state. A stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ is one that assigns a probability to each action given the current state.

In order to behave well, an agent should consider the long-term rewards of its actions, rather than only the immediate reward returned by $\mathcal{R}$. For example, in figure 2–1 the only time the agent receives a reward is when it reaches the goal state. By considering only the immediate reward, the agent would choose actions

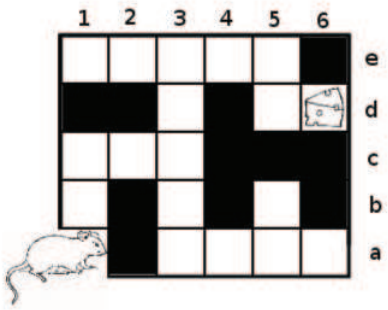

Figure 2–1: Grid-world example

randomly throughout the whole grid except for the states immediately next to the goal state. However, in state (3,d) the agent is clearly in a better position than in state (5,b), because at least one path to the goal from (3,d) is shorter than any path available from (5,b). This example suggests that states should have a value associated with them that captures the best possible behavior from a state.

The running time horizon of the agent is not generally specified beforehand, and a good decision-making method should handle horizons of arbitrary length, indeed, even of infinite length. Clearly, summing over all possible future rewards from a certain state can diverge if we consider an infinite horizon. For this reason, a discount factor $0 < \gamma < 1$ is introduced. This discount can be intuitively thought of as an interest rate in finance applications, and is meant to discourage rewards received too far in the future. For example, in the grid world of figure 2–1, introducing a discount factor pushes the agent to find the shortest path to the goal

from any state. The expected utility of an agent following a policy π is given by

$$\mathbb{E}_{\pi}^{\infty}[U] = \sum_{t=0}^{\infty} \gamma^t r_t^{\pi} < \infty. \tag{2.1}$$

where r_t^{π} is the expected reward received at time t when using policy π. An agent will try to find a policy maximizing equation (2.1) in order to achieve an optimal behavior. We immediately obtain the following inequality:

$$\mathbb{E}_{\pi}^{\infty}[U] \leq \frac{R_{max}}{1-\gamma} \tag{2.2}$$

for all policies π.

The Maximum Expected Utility principle (Russell and Norvig, 1995) states that a rational agent should choose the action that maximizes its expected utility. In an MDP, the maximum expected utilities from each state obey the Bellman optimality equations (Bellman, 1957) (note the recursive nature of this definition):

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

Inequality (2.2) also implies

$$V^*(s) \leq \frac{R_{max}}{1-\gamma}. \qquad \forall s \in \mathcal{S}$$

For convenience, denote $V_{max} = \frac{R_{max}}{1-\gamma}$. We may also wish to denote $V_{min} = \frac{R_{min}}{1-\gamma}$. By assumption 2.1.1 we have that $V_{min} = 0$, but we shall leave it as V_{min} for generality.

Similarly, one can define a set of state-action utilities, which quantify the value of taking each action from each state, and are defined as:

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') V^*(s'). \tag{2.4}$$

The solution of Equations (2.3) can be computed exactly and approximately, and we will review a number of these methods in the following sections. Note that if we have a solution to Equations (2.3), we can easily extract the optimal policy via the following equation:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left\{ \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') \cdot V^*(s') \right\} = \arg\max_{a \in \mathcal{A}} Q^*(s,a). \tag{2.5}$$

2.2 Computing the value function

In this section we present three standard methods for computing the value functions defined in Equations (2.3).

2.2.1 Value iteration

Bellman (1957) introduced Dynamic Programming (DP) as an approach to solving problems exhibiting a recursive nature as in equations (2.3). The basic idea is to start from an initial estimate V^0 of the value function and compute successive approximations via the following equation:

$$V^{n+1}(s) = \max_{a \in \mathcal{A}} \left\{ \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') V^n(s') \right\}. \tag{2.6}$$

Computation of the value function by this method of successive approximations is known as *value iteration*. The successive approximations V^n converge to

V^* in max norm, and value iteration finds a stationary policy that is near-optimal within a finite number of iterations (Puterman, 1994).

We restate a useful theorem from (Puterman, 1994):

Theorem 2.2.1. *Suppose* $0 \leq \gamma < 1$, $\mathcal{S}$ *is finite or countable, and* $\mathcal{R}(s,a)$ *is bounded. Then there exists a unique* V^* *satisfying equations (2.3).*

2.2.2 Policy iteration

The optimal policy π^* may not be very sensitive to the actual numerical values of the value functions (Russell and Norvig, 1995). In other words, the optimal policy may be reached before the value functions have converged to their fixed point (*i.e.* $V^n \neq V^*$ but $\pi^n = \pi^*$), where π^n is the policy extracted from V^n by equation (2.5). This suggests an alternate approach to obtaining the optimal decisions at each state, called *policy iteration*.

Given a certain deterministic policy π, we can compute the value of all the states when the agent is committed to π by the following set of equations:

$$V^\pi(s) = \mathcal{R}(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,\pi(s),s') V^\pi(s'). \tag{2.7}$$

Policy iteration works by starting from an initial policy π^0, computing V^0 via equations (2.7), and then extracting an improved policy π^1 from V^0 via $\pi^1 = \arg\max_{a \in \mathcal{A}} \left\{ \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') V^0(s') \right\}$ The algorithm continues until the policy no longer changes.

This is a sufficient stopping condition, since it has been shown that the sequence of value functions $\{V^n\}$ generated by policy iteration converges monotonically in max norm to V^* (Puterman, 1994).

2.2.3 Linear Programming

Finally, we present an alternate approach to computing the value functions defined by equations (2.3) which uses Linear Programming (LP). Linear programming is a technique developed by G.B. Dantzig in the late 1940's for solving optimization problems. Its elegant theory has been used in many fields (see (Chvátal, 1983)) and is well understood.

The Bellman optimality equations (2.3) can be rephrased as the following primal LP:

$$\begin{aligned} & \text{minimize} && \sum_{s\in\mathcal{S}} \alpha(s)V(s) \\ \text{such that} \quad & && V(s) - \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')V(s') \geq \mathcal{R}(s,a) \\ & && \forall s \in \mathcal{S}. \qquad \forall a \in \mathcal{A}. \end{aligned} \tag{2.8}$$

The vector α in the objective function can be considered as state-relevance weights. For finite MDPs (*i.e.* with finite state and action spaces) any strictly positive constants will produce the correct value function. For infinite state spaces (where the LP (2.8) becomes an infinite-dimensional LP) the choice of the α vector cannot be arbitrary. Indeed, it must be the topological dual of the primal variable space in order to guarantee continuity of the objective function and the constraints (see (Anderson and Nash, 1987)). We shall deal with this issue in section 6.1.

The above primal LP can be written in its dual form, yielding the dual LP:

$$\begin{aligned} & \text{maximize} && \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}}\mathcal{R}(s,a)x(s,a) \\ \text{such that} \quad & && \sum_{a\in\mathcal{A}}x(t,a) - \gamma\sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}}\mathcal{P}(s,a,t)x(s,a) = \alpha(t) \qquad (2.9) \\ & x(s,a) \geq 0 && \forall a \in \mathcal{A}.s \in \mathcal{S} \end{aligned}$$

The value of the decision variables $x(s,a)$ in the dual LP have an intuitive description. They represent the total discounted joint probability under initial-state distribution α that the system occupies state s and chooses action a (Puterman, 1994).

Since finite MDPs yield a finite LP we have no duality gap, thus, the value of the primal and the dual LP are the same. Furthermore, the value function computed by the LP is equal to the fixed point solution to equations (2.3) (Puterman, 1994). Note that the primal LP has $|S|$ variables and $|S| \times |A|$ constraints, whereas the dual LP has $|S| \times |A|$ variables and $|S|$ constraints. Thus, for finite MDPs, it is preferable to solve the dual LP.

2.3 Approximating the value function

The methods presented in section 2.2 are suitable for MDPs whose state and action spaces are not very large. If the MDPs have a very large state or action space, direct computation with the above methods is infeasible. In this section we briefly review some methods for approximating the value function when the size of the MDP makes exact methods infeasible.

2.3.1 Function approximators

In supervised learning, function approximation refers to a wide variety of techniques for constructing a representation based on data given in the form of inputs and desired real-valued outputs. This approach can be "imported" in MDPs by treating the state as an input and providing an output based on equation (2.3), for instance. This approach has been widely studied and used for MDPs using various techniques such as superimposed *tilings* over the state space (Albus, 1981), linear combinations of basis functions (Cherkassky and Mulier, 1996), radial basis function networks (Sutton and Barto, 1998), and many more (see (Ratitch, 2004) and (Sutton and Barto, 1998) for a survey of various function approximation methods). We do not focus on function approximation in this thesis, but mention them because one approach is closely related to some of the methods we present further on.

2.3.2 Linear Programming Approximations

The idea of Linear Programming Approximation (LPA) is to choose a set of basis functions that would span most of the region of interest in a large state space. The value function would then be approximated by a linear combination of the basis functions. The coefficients for each basis function are found by means of an linear optimization solver.

When considering the primal LP formulation (see equation (2.8)) of the value function, by using a set of basis functions to span the state space, the number of variables is reduced to the number of basis functions chosen. This idea was introduced in (Schweitzer and Seidmann, 1985) and further developed in (de Farias

and Roy, 2003; Hauskrecht and Kveton, 2004; Guestrin et al., 2002). Although LPAs have proven to be quite effective in practice, their success is directly affected by the choice of basis functions, which is not evident.

Furthermore, even though the number of variables can be reduced in this way, the number of constraints still remains arbitrarily large. To overcome this, de Farias and Roy (2001) propose sampling from the constraints in order to reduce the size of the constraint space. Unfortunately their method is not very practical, as their theoretical results rely on prior knowledge of the optimal policy, which in general is not known. The authors propose some heuristics to bypass this requirement, but at the expense of the theoretical guarantees.

2.3.3 Sparse Sampling

In (Kearns et al., 1999), the authors present an approximation for large state spaces using statistical sampling. Their algorithm requires a generative model that receives an input state and action and outputs the next state according to the MDP transition model. The idea is to start from some state in the MDP and create a sparse tree from it. From any state at level h in the sparse tree, a predefined number of transitions C are sampled for each action. This set of transition samples creates the $h + 1$ level in the sparse tree. The process is continued until a desired depth H. The authors present an analysis to determine the values of C and H that would yield a value function that is within ϵ of the optimal value function. Their algorithm is independent of the size of the state space, depending solely on V_{max}, γ and the desired precision ϵ. Thus, their algorithm is able to handle problems with an infinite state space.

Although theoretically satisfying, the required values for C and H make the resulting sparse MDP infeasible in practice. Nevertheless, the ideas can be applied with less conservative values for these parameters. This thesis will rely heavily on this idea.

2.4 Factored Markov Decision Processes

Many real world problems enjoy an underlying structure in the state space that allow one to express transition probabilities more compactly, by using state variables. The different combinations of all the values of these state variables produce all the possible states in the system. A standard example is the coffee robot problem (Boutilier et al., 1995), in which a robot is to get coffee for a user and avoid getting wet. The user is at the office and the coffee must be purchased at the store. Since going to the store means going outside, there is a risk of getting wet that is dependent on the probability of rain. The robot can choose from four actions: move (between the office and the shop), buy coffee, deliver coffee, and get umbrella. The state space is represented by the Boolean variables *location, hasRobotCoffee, hasUserCoffee, hasUmbrella, isRaining* and *isWet*. The different instantiations of the 6 state variables produce the 64 distinct states. In general, state variables need not be Boolean. Note that the number of distinct states increases exponentially with the number of state variables.

Independence relations can be exploited in order to provide a compact representation of the transition probabilities, instead of explicitly enumerating all states. In the coffee robot example, for instance, the value of state variable *hasRobotCoffee* at time step t does not affect the value of state variable *isRaining*

at time step $t + 1$. In fact, *isRaining* is independent of all other state variables except itself.

This suggests an alternate form of representing the transition probabilities: using *Dynamic Bayesian (or Belief) Networks* (DBNs) (Dean and Kanazawa, 1989). A Bayesian network is a directed, acyclic graph (DAG) where the nodes are the state variables, and the arrows represent *influence* (*i.e.* an arrow from node X to node Y indicates that the value of node X has a direct influence on the value of node Y). At each node X a conditional probability table (CPT) is stored that contains the probabilities of the different values for X given the values of its parents (the nodes $\{Y\}$ with an arrow to X).

A Dynamic Bayesian Network (DBN) is Bayesian Network with two copies of all the state variables, for two successive time steps t and $t + 1$. The directed arrows go from the variables at time step t to the ones at time step $t + 1$ (since we are assuming the Markov property).

A factored MDP model will have one DBN for each action a (Koller and Parr, 2000). An example of the representation for the robot coffee domain is depicted in figure 2–2, where the left column holds the variables at time step t and the right column holds the variables at time step $t + 1$ (we have omitted the CPTs for clarity). The size of the representation is dependent on the number of dependencies between state variables. Indeed, if every state variable is affected by the value of all the state variables in the previous time step, then the DBN representation is no smaller than the tabular representation.

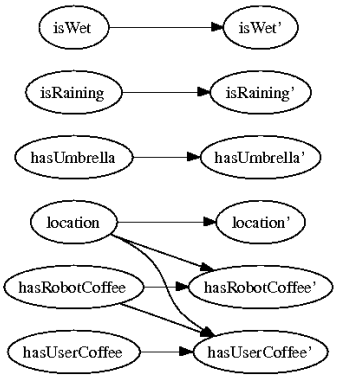

Figure 2–2: Coffee robot example: DBN for deliver coffee action

The reward function can also benefit from the structure of the state space if it can be decomposed into localized additive reward functions depending on a subset of the state variables $\mathbf{X}$.

Unfortunately, the value function for factored MDPs usually does not exhibit any factored structure, and so a compact representation is not possible without approximations.

2.4.1 Algebraic Decision Diagrams

Algebraic Decision Diagrams (ADDs) are one such structure which have been widely used to represent factored MDPs. To describe them we must first describe Binary Decision Diagrams (BDDs).

BDDs are a compact way of representing Boolean functions $f : \mathbb{B}^n \mapsto \mathbb{B}$ in a tree-like fashion (Bryant, 1986). Consider ordering the binary variables as $b_0, b_1, \cdots, b_n$ and constructing a complete binary tree in the following manner. At level i there are only nodes representing variable b_i. Each node has two branches stemming from it into level $i + 1$, representing the two possible Boolean

values. A traversal of a branch of the binary tree yields an instantiation of all the variables, and thus the leaf contains the result of the function f applied to such an instantiation. This simple construction may not be very efficient, as there may be many identical subtrees. BDDs take advantage of this and reduce a full binary tree into a more compact form by merging any identical subtree and eliminating any node whose two children are identical. Consider the Boolean function $x_1 \vee (x_2 \wedge x_0)$ over three variables. Figure 2–3 (a) illustrates the complete binary tree for this Boolean function, and figure 2–3 (b) demonstrates the reduction obtained through the BDD construction. In all the BDD diagrams below a solid line represents the *true* branch while the dotted line represents the *false* branch.

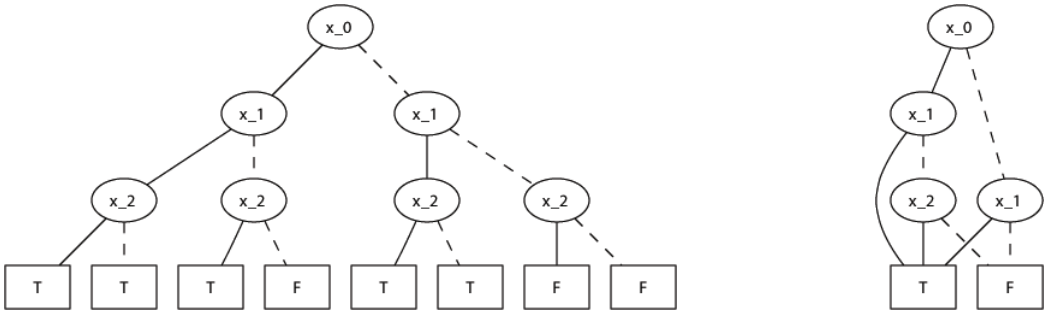

Figure 2–3: Example Boolean function: (a) Full binary tree representation; (b) BDD representation

ADDs generalize BDDs in that the value at the leaves need not be Boolean. In other words, ADDs are a compact way of representing real valued functions over Boolean variables $f : \mathbb{B}^n \mapsto \mathbb{R}$ (R.I. Bahar et al., 1993). Furthermore, ADDs can be used to represent real valued functions over variables which take on more than two values. This can be done by considering the binary representation of each multi-valued variable and assigning a binary variable to each bit of such a

representation. ADDs are then a compact way of representing real valued functions over variables that can take on a finite set of values.

It should be noted that the size of the resulting BDD or ADD is dependent on the ordering of the variables chosen. Figure 2–4 demonstrates two BDDs representing the same Boolean function as above but with different variable orderings. Finding the optimal ordering of the variables for a BDD (or ADD) is actually an NP-hard problem, and as such, the designer must rely on heuristics to be able to compactly represent a function.

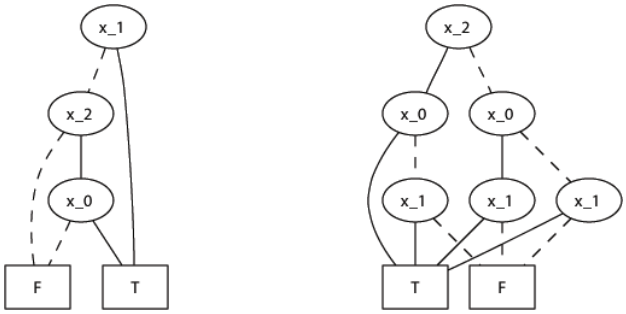

Figure 2–4: BDD orderings for example Boolean function: (a) good ordering; (b) bad ordering

Stochastic Planning Using Decision Diagrams (SPUDD) is a planning algorithm introduced in (Hoey et al., 1999), which uses ADDs to encode a factored MDP and performs a form of value iteration performs a form of value iteration solely through operations on these ADDs. Their experimental results show that in many cases there are big savings when using SPUDD for value iteration, rather than standard value iteration. The implementation of SPUDD is based on CUDD (Colorado University Decision Diagrams) (Somenzi, 1998), which is a library of C routines designed to manipulate ADDs.

2.5 Reinforcement Learning

The previous sections presented methods for solving MDPs where the model is known. In many cases, however, this is not the case. In these situations, the agent must *learn* the model and how to act by interacting with the environment. The agent must adapt its behavior in order to maximize the expected utility. In order to have an idea of the utility of each choice the agent must try all the actions from all the different states.

Reinforcement Learning (RL) is the field of Machine Learning that deals with this learning problem. There is a vast array of algorithms designed for this learning problem (see (Sutton and Barto, 1998) for an excellent survey of available methods) which can be classified into different types of methods. We will only review some of them and point the reader to (Sutton and Barto, 1998) for more information.

Model-based methods are methods which maintain an explicit representation of the model estimate throughout the learning process. As such, the model estimate at any point can be considered as a known MDP and any of the methods described in section 2.2 may be used to obtain a value function estimate. One advantage of this type of method is that if any prior knowledge of the model is available, it may be used as the initial model estimate and may shorten the learning time.

Model-free methods do not maintain a model estimate, but rather update the value function estimates directly and use these estimates to make the action choices. One advantage of these methods is that they avoid having to store the

immediate rewards and transition probabilities. However, prior knowledge of the dynamics of the system does not provide an improved starting point.

One well-known type of model-free method is Q-learning (which is in fact a type of Temporal Difference learning (Sutton and Barto, 1998)). Q-learning maintains an estimate of the action value functions (see equation (2.4)) by performing an update after each time step of agent-environment interaction. After observing a transition from state s to state s' under action a and receiving reward r, the agent performs the following update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]. \tag{2.10}$$

where $\alpha \in (0,1)$ is the learning rate. The learning rate can be thought of as the step size in standard gradient descent methods (see (Duda et al., 2000)).

The action values maintained by Q-learning are guaranteed to converge to the optimal action value function if all actions are tried from all states infinitely often. This algorithm works well in practice, and its ease of implementation makes it an enticing candidate for RL agents.

2.5.1 Exploration

An agent is faced with a conundrum when posed with this sort of problem. On the one hand, it needs to *explore* its environment in order to obtain valuable experience which will allow it to better understand the dynamics of the environment. On the other hand, it needs to *exploit* whatever knowledge of the environment it already has in order to maximize the reward received. Exploring is

expensive and risky, as the agent may choose sub-optimal or even dangerous actions; however, a lack of exploration may lead the agent to be stuck with a policy that is sub-optimal (local maxima). This is known as the *exploration problem*, and is the main problem we will address in this thesis.

Exploration has been studied extensively from a theoretical point of view using bandit problems. A *one-armed bandit* is a slot machine where a user can insert a coin, pull the handle and collect any earnings. An n-armed bandit is a slot machine with n handles. The user inserts a coin and must choose which handle to pull. Each handle obeys a different probability distribution that prescribes the expected earnings when playing it. If the model is known beforehand the problem is trivial, as the user need only play the handles with the highest expected earnings. If the model is not known, the user must maintain estimates of the expected earnings of each handle. A *greedy* policy is one that always chooses the handle with the highest current estimate of the expected return. An agent following a strictly greedy policy is only exploiting but not exploring. When the agent chooses a handle that is not the optimal given the current estimate, the agent is exploring.

For bandit problems, the balance between exploration and exploitation has been solved exactly in (Gittins and Jones, 1979). Their method computes *Gittins indices* for each handle which depend on the current model estimate. Gittins indices map each available handle to a real number in such a way that an optimal policy consists of always choosing the handle with the highest index. This method

is not directly applicable to general MDPs, as it relies on the fact that in bandit problems there is only one state.

2.5.2 ϵ-greedy and Boltzmann exploration

Two simple and well-known methods for exploration in MDPs are ϵ-greedy and Boltzmann methods. In ϵ-greedy exploration, the agent chooses actions greedily (based on the current value function estimate) with probability $1 - \epsilon$, and chooses actions randomly with probability ϵ. Most implementations decay the value of ϵ throughout time. This yields an agent that, the more knowledge it has about the environment, the less it explores.

Boltzmann exploration use a Gibbs or Boltzmann distribution over the available actions. The Boltzmann distribution comes from statistical mechanics and is used in physics and chemistry to determine the percentage of molecules within a certain velocity range given the temperature of the system. In this context, the Boltzmann distribution defines the probability of choosing action a from state s by:

$$p(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in \mathcal{A}} e^{Q(s,a')\tau}} \tag{2.11}$$

where τ is the temperature parameter. A high temperature will yield a uniform distribution over all actions (and thus promote exploration), while a low temperature will choose actions with high expected value with much higher probability. As in ϵ-greedy methods, τ is normally decreased over time.

2.5.3 Interval Estimation methods

The two methods presented in section 2.5.2 choose actions randomly while exploring. Interval Estimation (IE) attempts to overcome this problem by computing confidence intervals for the action values (Kaelbling, 1993). It always selects the action with the largest upper interval boundary, thereby giving an advantage to actions where there is a large uncertainty.

Since in RL one is concerned with the uncertainty in the model, Wiering and Schmidhuber (1998) introduced Model based IE, where confidence intervals are maintained on the probability distributions for each state-action pair. Like IE, it gives an advantage to state-action pairs that have not been explored sufficiently. This idea was further developed and analyzed in (Strehl and Littman, 2004) and (Strehl and Littman, 2005), where formal learning time guarantees are presented.

2.5.4 E^3 and variants

In (Kearns and Singh, 1998) the E^3 (Explicit Explore or Exploit) algorithm is introduced. The main idea of the algorithm is to divide the state space into *known* and *unknown* states. When in an unknown state, the algorithm is in an exploratory phase where its policy is guaranteed to try to increase the agent's knowledge of the environment; when in a known state, the algorithm is in an exploitative phase where it chooses actions greedily according to its model estimate. The model estimate is used only on a subset of the state space (namely, the known states). All states start off being unknown, and a state becomes known after it has been visited at least a predetermined number of times M. The value of M

is computed as a function of $1/\epsilon$, $1/\delta$, $|\mathcal{S}|$ and R_{max}, where ϵ is the desired precision in the value function and δ is the desired confidence in the value estimates. The authors present bounds on the number of samples needed to obtain a good approximation of the optimal value function.

Despite the strong theoretical guarantees presented in (Kearns and Singh, 1998), the implementation of E^3 is not straightforward and makes it hard to use. In (Brafman and Tennenholtz, 2001) the authors present R-MAX, an algorithm which is simpler and generalizes E^3. It creates an absorbing state s_0 which returns a reward of R_{max} upon entering it (hence the name of the algorithm). The algorithm maintains two models, one for planning and another for estimation. The model for estimation is updated after every transition. The planning model is used to choose the next action. Initially, all states transition deterministically to s_0 under every action in the planning model. Once a state has been visited at least M times, it becomes known and the distribution from the estimation model is used for planning from that state.

For factored MDPs E^3 is not sufficient, as it maintains counts for states and thus bypasses the structure in the state space. To overcome this, (Kearns and Koller, 1999) present DBN-E^3, an algorithm similar to E^3, except that instead of states being known or unknown, it is the CPT entries in the DBNs that can be known or unknown. The authors present similar bounds guaranteeing the efficiency and optimality of their algorithm. Similarly, R-MAX has been generalized for factored MDPs (Guestrin et al., 2002)

Note that all the algorithms presented in this section still require a planning algorithm to choose the next action given the model. This is left to the designer to choose, although in the general case, an optimal planner may not be available for large domains. In (Guestrin et al., 2002) the authors present a modification of Factored R-MAX using Approximate Linear Programming. Their methods are able to handle large domains while still maintaining strong theoretical guarantees. However, a good choice of basis functions is critical for good results.

2.6 Summary

In this chapter the necessary background for the remainder of this thesis has been presented. MDPs were introduced as the mathematical model for planning in stochastic environments, along with various methods for computing the value function and optimal policy. Some approximate methods were presented for larger domains where exact methods become infeasible. Observing that in many problems the state space enjoys an underlying structure, we presented factored MDPs, and an efficient value iteration algorithm (SPUDD).

When the model of the MDP is unknown, the agent needs to interact with the environment to learn the model and the optimal behavior. Reinforcement Learning is the field which studies these types of learning problems. We introduced various types of RL methods, along with the exploration problem in RL. Exploration is the main focus of this thesis, and thus, we finished the chapter presenting various algorithms that produce different forms of exploration, with different theoretical guarantees.

Note that our discussion here does not include heuristic methods for directing exploration, as many of these are ad-hoc and not directly related to our work. Bayesian methods, which are more closely related to our work, will be presented in detail later.

CHAPTER 3
Bayesian Learning

In Bayesian learning the agent maintains and updates a model estimate, starting from a *Bayesian prior*. A prior encodes any initial information about the environment the agent may have. This prior information can capture various forms of knowledge and can come from various sources. For example, a design engineer could encode a rough estimate of the transition probabilities based on her experience. A robot could have a rough estimate of its dynamics from previous runs in a somewhat similar domain. The set of states $\{s'\}$ that are reachable from a state s under action a could be known in advance such as in grid world domains, where the immediate neighbors of s are the only reachable states under any action.

Bayesian learning attempts to maintain a model estimate, update it accordingly after any observation, and use the current model estimate to make a (near) optimal action choice in the next time step. A formal description of this idea is presented in the next section, followed by related work.

3.1 Hyper MDPs

3.1.1 Intuitive description

Bayesian learning in MDPs can be traced back to (Bellman and Kalaba, 1959), where *Adaptive Control Processes* are introduced. Informally, this model pairs a current model estimate (the information state) with the current state in the MDP (the physical state) to create a *hyper state*. A *Hyper MDP* is then created

by allowing the transition probabilities between hyper states to be determined by the current model estimate at the hyper state. [1] Heretofore *MDP* will refer to the original MDP (whose model is unknown) and *Hyper MDP* will refer to the constructed Hyper MDP.

Note that since we are using the model estimate at each hyper state to determine the possible transitions, the model of the Hyper MDP is known beforehand. It has been shown that if the agent considers all the possible hyper states it could reach and computes the value function for all these hyper states, this value function produces an optimal exploration strategy.

An intuitive way to think of a Hyper MDP is as a tree. The root r is the hyper state defined by the physical state s the agent is in and the current model estimate ψ. By considering all possible transitions under all possible actions and the resulting updated model estimates, the agent can construct the next-level set of hyper states. Continuing in this fashion one can see that the tree is in fact infinite (since we can continue interacting with the environment and updating the model estimate based on the observations indefinitely). Also note that the model gets *refined* the further one goes down the tree. Note that the data received will determine exactly one path that the agent takes in the infinite tree. With this graphical representation in mind, we will heretofore refer to this model as the

[1] In (Bellman and Kalaba, 1959) they use the terms 'generalized state' and 'Adaptive Control Process' to refer to what we call 'hyper state' and 'Hyper MDP', respectively.

Hyper MDP or the hyper tree, depending on which facilitates the exposition of ideas.

3.1.2 Formal definition

We will follow the notation of (Martin, 1967) for the formal definition of Hyper MDPs (*i.e.* Adaptive Control Processes). Since we will be using our model estimate to obtain a probability distribution and we will be updating this model estimate as we interact with the environment, we need to consider distributions from a family $\mathcal{H}$ that are closed under updates and which can be indexed by $\psi \in \Psi$, where Ψ is the admissible set of parameters for $\mathcal{H}$. The different members of $\mathcal{H}$ differ only in the values assigned to ψ. In other words, ψ represents our information state, and encodes the prior and any updates performed from interactions with the environment. Thus, our hyper states are of the form $(s, \psi) \in \mathcal{S} \times \Psi$.

If we assume finite state and action spaces, the uncertainty in the transition matrix can be denoted by a random $|\mathcal{S}| \times |\mathcal{A}|$ tensor $\mathcal{P}$, with a prior probability distribution function $H(\mathcal{P}|\psi) \in \mathcal{H}$. We may define the set of all $|\mathcal{S}| \times |\mathcal{A}|$ generalized stochastic tensors by:

$$\mathcal{M} = \left\{ \mathcal{P} \quad | \quad \mathcal{P} \text{ is } |\mathcal{A}| \times |\mathcal{S}|, \mathcal{P}(s,a,s') \geq 0, \qquad \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') = 1 \quad (a \in \mathcal{A}; s, s' \in \mathcal{S}) \right\}$$

The range of $H(\mathcal{P}|\psi)$ is $\mathcal{M}$, and we have that

$$\int_{\mathcal{M}} dH(\mathcal{P}|\psi) = 1.$$

We can now extract the transition probabilities given an information state ψ. The probability of ending in state s' given that action a is chosen and the agent is in hyper state (s, ψ) is given by

$$\bar{p}^a_{s,s'}(\psi) = \int_{\mathcal{M}} \mathcal{P}(s, a, s') dH(\mathcal{P}|\psi). \tag{3.1}$$

From this we can determine the expected reward when choosing action a in hyper state (s, ψ) by

$$\bar{r}^a_s(\psi) = \sum_{s' \in \mathcal{S}} \bar{p}^a_{s,s'}(\psi) \mathcal{R}(s, a, s'). \tag{3.2}$$

Note that equation (3.2) assumes that the reward function is known, which in general may not be the case. If the reward function is not known, equation (3.2) becomes:

$$\bar{r}^a_s(\psi) = \sum_{s' \in \mathcal{S}} \int_{C_b[0, R_{max}]} \bar{p}^a_{s,s'}(\psi) d\mu(\psi). \tag{3.3}$$

where $C_b[0, R_{max}]$ is the space of all bounded continuous measurable reward functions on $[0, R_{max}]$, and μ is a measure over $C[0, R_{max}]$ given the information in ψ. In this thesis, however, we will assume that the reward function is known beforehand.

Let $D : \Psi \mapsto \Psi$ be an operator that updates our distribution parameter (or information state) based on observations experienced, and let $D^a_{s,s'}(\psi)$ denote the update based on a transition observation from state s to state s' under action a. As was mentioned in section 3.1.1, we are concerned with computing the optimal value function over hyper states, $V^*(s, \psi)$. Given the formulation above, we can

restate the Bellman optimality equations (2.3) for Hyper MDPs:

$$V(s,\psi) = \max_{a\in\mathcal{A}} \left\{ \bar{r}_s^a(\psi) + \gamma \sum_{s'\in\mathcal{S}} \bar{p}_{s,s'}^a(\psi) V(s', D_{s,s'}^a(\psi)) \right\} \tag{3.4}$$
$$\text{where } (s,\psi) \in \mathcal{S}\times\Psi; \quad 0 \le \gamma < 1$$

In section 3.1.1 we mentioned that there are an infinite number of hyper states, and this fact is reasserted by the set of Equations (3.4). Indeed, it is not immediately obvious that the set of Equations (3.4) should admit an optimal solution, let alone a unique one. Fortuitously, the following theorem from (Martin, 1967) demonstrates that this is actually the case.

Theorem 3.1.1. *There exists a unique set of bounded functions* $\{v_i(\psi)\}$ *which satisfies the set of equations (3.4).*

Despite Theorem 3.1.1, the fact that there are an infinite number of hyper states implies that an exact computation of Equations (3.4) would be infeasible.

It would be convenient to deal with a finite subset of the Hyper MDP.
A commonly used approach is to truncate the unfolding of the value function computation necessary for solving Equations (3.4). By truncating at a certain depth n, we obtain a similar set of equations:

$$V^{n+1}(s,\psi) = \max_{a\in\mathcal{A}} \left\{ \bar{r}_s^a(\psi) + \gamma \sum_{s'\in\mathcal{S}} \bar{p}_{s,s'}^a(\psi) V^n(s', D_{s,s'}^a(\psi)) \right\}, \tag{3.5}$$
$$(s,\psi) \in \mathcal{S}\times\Psi; \quad n = 0,1,2,\cdots; \quad 0 \le \gamma < 1,$$
$$\text{where } V^0(s,\psi) = V_{max} = \frac{R_{max}}{1-\gamma}, \quad (s,\psi) \in \mathcal{S}\times\Psi.$$

The choice of $V^0(s,\psi) = V_{max}$ is specific to our approach. As was mentioned in section 2.1, V_{max} is an upper bound on the value function for any state. By truncating the infinite unfolding of the value function at a depth n, the choice of V^0 is an estimate of the true value function for the hyper states at the fringe of the truncation. Thus, by setting the *leaves* of the hyper tree to this upper bound, we are promoting exploration by being optimistic about the true value of these fringe hyper states.

Martin (1967) shows that Equations (3.5) can be used as approximations and converge monotonically to the solution of Equations (3.4). We will restate a theorem from (Martin, 1967) which will be necessary for our theoretical analysis in section 5.2. The theorem has been rephrased for our situation and is stated without proof.

Theorem 3.1.2. *Let $\{V^n(s,\psi)\}$ be a sequence of successive approximations defined by Equations (3.5). Then the error of the nth approximant has the bound*

$$|V(s,\psi) - V^n(s,\psi)| \leq \gamma^n V_{max}.$$

3.2 Choice of distribution

As mentioned in the previous section, we need to choose a distribution that is closed under updates and which can be indexed by a set of parameters ψ. The multinomial distribution is a natural choice for finite state and action spaces.

The multinomial distribution is essentially obtained from counts of observed transitions. Let the matrix M be a $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ matrix of observation counts, such that element $M^a_{s,s'}$ holds the number of times we have observed transition

$s \rightarrow s'$ under action a. From such a matrix we can easily extract the desired probabilities via:

$$\hat{P}(s, a, s') = \frac{M^a_{s,s'}}{\sum_{s'' \in \mathcal{S}} M^a_{s,s''}}.$$

Thus, the matrix M is our indexing parameter used to extract the necessary distributions. For generality, we will continue to refer to the indexing parameter as ψ. Note also that with the multinomial distribution it is easy to encode a prior by assigning the initial counts in M appropriately. Complete uncertainty of the domain can be setting M to the constant matrix 1. Events which are more likely to occur can be assigned a higher value, while events which are known to never occur can be set to zero.

3.3 Related work

We will mention some work on Bayesian exploration using the Hyper MDP model. In (Dearden et al., 1999) the authors present a method which uses the *value of perfect information* (Howard, 1966) to estimate the possible gains from exploration. The computation of the value of perfect information is clearly infeasible and the authors present various myopic estimates. This approach is similar to IE methods (see section 2.5.3) in that it attempts to give an advantage to exploratory actions.

In (Duff, 2002), various heuristic methods are presented for approximating an exact solution to the Hyper MDP. The methods include converting the problem into a local semi-Markov Decision Process (Puterman, 1994) by considering two possibilities from each physical state: either return to the same state or enter a 'sojourn' period through the rest of the states. Each action then becomes a

random process whose reward is determined by the length of sojourn. With this construction, Gittins indices may be computed and used for action selection. The author also considers approximating the value function over hyper states by parameterizing them as a linear combination of information state components. Finally, an approach is presented where the likelihood of visiting each hyper state is estimated by modeling information state components by diffusion processes and defining a system of flux constraints between connected hyper states to acknowledge interdependencies. These dynamics are formulated in terms of stochastic differential equations. Although the methods presented in (Duff, 2002) produce empirically good results, the author fails to provide any theoretical guarantees of their performance.

In (Wang et al., 2005) an algorithm based on the idea from (Kearns et al., 1999) is presented to approximate the value of the Hyper MDP. They construct a sparse tree from the full Hyper MDP, using the information states ψ as a generative model. In contrast to (Kearns et al., 1999), they do not expand all actions in their construction. Instead, they use Thompson sampling (Thompson, 1933) to pick their actions. At each hyper state (s, ψ), the optimal action value function is computed for the MDP with model given by ψ. This estimate is then used to choose the optimal action(s), and only these action(s) are expanded. Finally, any actions that have not been expanded are still evaluated by multiplying the expected immediate reward when performing the action by the number of decisions remaining to the selected horizon.

The idea in (Wang et al., 2005) serves as the inspiration for the algorithm presented in the next chapter.

3.4 Summary

In this chapter we have introduced Hyper MDPs, which will be used extensively throughout this thesis. We presented certain theorems from (Martin, 1967) which will be necessary later for the theoretical analysis. Finally, we presented some related work on exploration using the Hyper MDP model. In particular, the work from (Wang et al., 2005) is closely related to the approach we present in the next chapter.

CHAPTER 4
Using Linear Programming for Bayesian learning

The idea of sampling a finite subtree of the hyper tree, as in (Wang et al., 2005), is promising. This method provides direct control over how large we would like the tree to grow. However, the use of Thompson sampling in the algorithm of (Wang et al., 2005) has a major disadvantage. At each hyper state, (s, ψ), the value function for the original MDP, using the model indexed by ψ, is computed in order to determine the best action choice from s. If the original MDP is large, this computation is very expensive. To perform this computation on *each* hyper state greatly limits the size of the problems that this method can handle.

Furthermore, by sampling based on the model at the current hyper state, the sampling is somewhat myopic in that it assumes that the information state ψ does not change from then on. If the agent begins with a bad prior this would "blind" it from expanding other areas in the hyper tree.

We propose a new approach that attempts to improve the approach of (Wang et al., 2005). We present empirical evidence that our approach exhibits better performance than (Wang et al., 2005). This work has been previously published in (Castro and Precup, 2007).

4.1 The Approach

Our approach differs from that of (Wang et al., 2005) in four important ways. The first difference is that rather than computing the value function at each

hyper state to sample the next action, we choose the action uniformly randomly from the available actions. The sparse sampling algorithm introduced in (Kearns et al., 1999) (on which the approach in (Wang et al., 2005) and our approach are based) expands all available actions rather than sampling from the available actions. Our approach lies somewhere in between expanding all actions (as in (Kearns et al., 1999)) and using Thompson sampling (as in (Wang et al., 2005)). In contrast to (Wang et al., 2005), we fix the depth of the sampled Hyper MDP. Thus, by choosing an action uniformly randomly at each hyper state, the fixed horizon allows many actions to be sampled per hyper state. As we increase the maximum number of samples, we expand more actions per hyper state, bringing our algorithm closer to the original idea of (Kearns et al., 1999). This approach would create a more balanced tree in cases where Thompson sampling would create a tree with few, long trajectories.

The second difference is that we compute the value of the sampled hyper states using Linear Programming. In a manner similar to what was presented in (Puterman, 1994), the optimality equations for the hyper MDP can be formulated as a linear program as follows:

$$\begin{aligned} & \text{minimize} && \sum_{(i,\psi)\in\mathcal{S}\times\Psi} V(i,\psi) \\ \text{such that} \quad & && V(i,\psi) - \left[\bar{r}_i^a(\psi) + \gamma \sum_{j\in\mathcal{S}} \bar{p}_{ij}^a(\psi) V(j, D_{ij}^a(\psi))\right] \geq 0 \qquad (4.1) \\ & \forall (i,\psi) \in \mathcal{S}\times\Psi. && \forall a \in \mathcal{A}. \end{aligned}$$

We will refer to equations (4.1) as the *exact LP*. Note that the exact LP is a countably infinite dimensional LP because there are countably many distinct states and countably many constraints (see (Anderson and Nash, 1987)). We will obviously not attempt to solve the exact LP, but will construct a finite subset of the exact LP by sampling from the hyper tree. In particular, we will consider the finite subsets $K \subset \mathcal{S} \times \Psi$ and $B \subseteq \mathcal{A}$. The LP we will solve (heretofore referred to as the *sampled LP*) is then defined as:

$$\begin{aligned} & \text{minimize} \qquad \sum_i V(i,\psi) \\ \text{such that} \qquad & V(i,\psi) - \left[\bar{r}_i^a(\psi) + \gamma \sum_{j \in \rho_1(K)} \bar{p}_{ij}^a(\psi) V(j, D_{ij}^a(\psi)) \right] \geq 0 \qquad (4.2) \\ & \forall (i,\psi) \in K. \qquad \forall a \in B. \end{aligned}$$

Note that we make use of a projection operator $\rho_1 : \mathcal{S} \times \Psi \mapsto \mathcal{S}$. The choice of using Linear Programming was motivated by the success of Linear Programming Approximation techniques (see section 2.3.2) and the hope that similar techniques could be applied to this problem.

The third difference is that we introduce a *rest* parameter, whose purpose is to speed up the algorithm. Since we have opted to maintain a Dirichlet distribution with counts of observed transitions, at each iteration we only increase one of the counts in our matrix by one. It is then clear that the distribution does not change very much after a single transition. Thus, rather than constructing a new Hyper MDP after each iteration, we allow the *rest* parameter to determine how many iterations should pass before a new Hyper MDP is constructed.

Finally, the fourth difference is that we maintain action-value estimates for the state-action pairs of the original MDP, and update them using standard Q-learning (see section 2.5). We will use these action-value estimates for choosing actions during the *rest* period. We use this approach because Q-learning is simple and cheap, yet it often provides a good policy quickly.

4.2 The Algorithm

We will now detail our approach, which we will call the LP approach. Given an initial prior for our model, we sample a finite Hyper MDP as follows. At each hyper state (s, ψ), we choose an action a uniformly randomly from the available actions. We then extract the model from ψ to sample a transition $s \xrightarrow{a} s'$ leading us into a new hyper state $(s', D^a_{s,s'}(\psi))$. This is continued until the desired horizon is reached, at which point the sampling begins from the root of the hyper tree. Once the number of sampled hyper states (internal or fringe) reaches the maximum number of samples, we stop.

Note that there are many subtrees of the infinite hyper tree that are not sampled. Nonetheless, we need an estimate of these subtrees in order to solve the sampled LP. In our approach we assign the root of all unsampled subtrees a value of V_{max}. As explained in section 3.1.2, this is done to promote exploration by being optimistic about unsampled regions.

The algorithm is described in detail below. Algorithm *LPapproach* receives five parameters, whose type is given by $s \in \mathcal{S}; \psi \in \Psi; depth, maxSamples, rest \in \mathbb{N}$. For all our experiments we used CPLEX as our LP solver.

Algorithm 1 LPapproach$(s, \psi, depth, maxSamples, rest)$

$Q(s,a) \leftarrow 0. \qquad \forall (s,a) \in \mathcal{S} \times \mathcal{A}$
repeat
  $sampledLP \leftarrow sampleHyperMDP((s,\psi), depth, maxSamples)$
  Solve $sampledLP$ to obtain a value function $\bar{V} : K \mapsto \mathbb{R}$
  Choose an action by $a = \arg\max_a \left[\bar{r}^a_s(\psi) + \gamma \sum_{s' \in \rho_1(K)} \bar{p}^a_{s,s'}(\psi) \bar{V}(s', D^a_{s,s'}(\psi))\right]$
  Observe transition $s \xrightarrow{a} s'$
  $Q(s,a) \leftarrow Q(s,a) + \alpha[\bar{r}^a_s(\psi) + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
  $\psi \leftarrow D^a_{s,s'}(\psi)$
  $s \leftarrow s'$
  for $i = 1$ to $rest$ **do**
    Construct the following value function $\hat{V} : \mathcal{S} \mapsto \mathbb{R}$:

$$\hat{V}(s) = \left\{ \begin{array}{ll} \bar{V}(s,\psi) & \text{if } (s,\psi) \in K \\ \max_a Q(s,a) & \text{otherwise} \end{array} \right\}$$

    Choose an action by $a = \arg\max_a \left[\bar{r}^a_s(\psi) + \gamma \sum_{s' \in \mathcal{S}} \bar{p}^a_{s,s'}(\psi) \hat{V}(s')\right]$
    Observe transition $s \xrightarrow{a} s'$ and reward r
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $\psi \leftarrow D^a_{s,s'}(\psi)$
    $s \leftarrow s'$
  end for
until done exploring

The next algorithm defines $sampleHyperMDP$ (which is called by $LPapproach$) and is responsible for generating the sampled LP in the form of Equations (4.2). It works by sampling trajectories of the specified depth and choosing actions uniformly randomly at each hyper state. Each previously unsampled hyper state in the trajectory is added as a variable to the sampled LP (and thus, to K) until the specified maximum number of samples is reached. We use ζ to denote the set of states in the sampled trajectory.

Algorithm 2 sampleHyperMDP($(s, \psi), depth, maxSamples$)

1: $samples \leftarrow 0$
2: $K \leftarrow \emptyset$
3: **while** $samples \leq maxSamples$ **do**
4: Sample a trajectory of length $\min(depth, maxSamples - samples)$ from the hyper tree, choosing actions uniformly randomly at each hyper state. Let ζ be the set of states along the trajectory.
5: $samples \leftarrow samples + |(\zeta - K)|$
6: $K \leftarrow K \cup \zeta$
7: **end while**
8: Construct $sampledLP$ from K as defined in Equations (4.2)

The length of the trajectory in line 4 ensures that the number of samples taken does not exceed the specified maximum number of samples.

4.3 Problem domains

We will now present three problem domains on which the LP approach was tested. The three problems are of differing sizes and nature, in order to demonstrate the versatility of the LP approach.

4.3.1 Small MDP

The first problem is a two-state, three-action MDP, whose dynamics are illustrated in figure 4–1. The circles depict the states and the labeled arrows between them are the transition probabilities under the specified action. Action 1 is displayed on the top left, action 2 on the top right, and action 3 at the bottom. The reward received upon entering each of the two states is displayed inside the circle for each action. The agent always begins in the left state for all trials. Intuitively, the best policy is to perform action 1 until the right state is reached, and then do action 3 to remain in the right state.

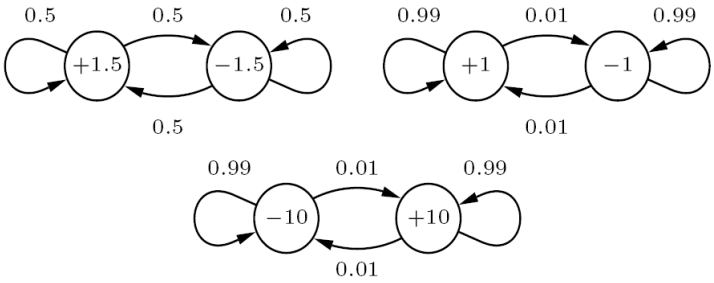

Figure 4–1: Dynamics of a two-state, three-action Small MDP

4.3.2 Bandit problem

The second problem is similar to the well known bandit problem (see section 2.5.1) with two slot machines (bandits) and three actions. In addition to the three actions, the model has three states. The dynamics of this model are illustrated in figure 4–2. Action 1 (don't gamble) is displayed as a solid black line, action 2 (gamble on machine 1) as a dashed line, and action 3 (gamble on machine 2) as a solid grey line. The leftmost state corresponds to not winning anything, the middle state corresponds to winning under machine 1 and the rightmost state corresponds to winning under machine 2. Unlike most slot machines, we have not associated any cost for gambling. Once again, the labels on the arrows depict the transition probabilities under each of the three actions. Naturally, the agent always begins in the leftmost state.

4.3.3 GridWorld

The third problem is a 2x4 grid world (*i.e.* 8 states) with a reward of +1 in the top right state and 0 everywhere else. There are four actions (North, South, East and West) with a 0.1 probability of remaining in the same state. If the agent

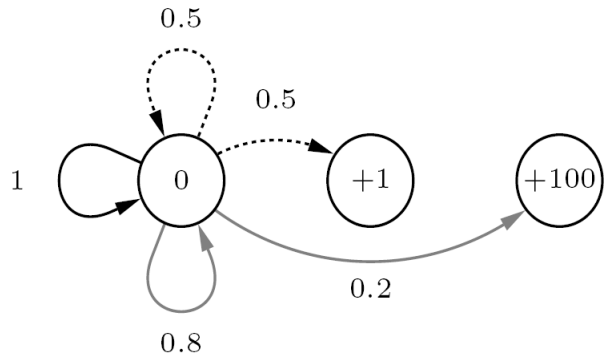

Figure 4–2: Dynamics of the bandit problem

tries to move into a wall it will deterministically stay in the same state. The agent always begins in the bottom left state. This problem is illustrated in figure 4–3.

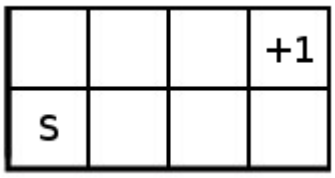

Figure 4–3: Dynamics of the grid world problem

4.4 Experimental results

In this section we present the experimental results on the three domains described in section 4.3. For all domains, the results were averaged over 30 independent runs. The graphs present the performance of our algorithm *as it is exploring*.

We ran the LP approach against four different types of algorithms. This first is a Q-learning agent (with $\alpha = 0.1$ [1]) using an ϵ-greedy approach (setting ϵ to $0.01, 0.1$, or 0.5). The second type is the algorithm described in (Wang et al., 2005). We also compared against a myopic agent (acting only based on immediate reward) and an agent which chooses actions randomly. These last two algorithms fared far worse than all the others, and so are omitted from our graphs for clarity. Finally, we also implemented an agent using Boltzmann exploration, but the results (for varying temperature values) was similar to the results for ϵ-greedy, and so the results are omitted from our graphs for clarity.

For the LP approach we experimented with different parameter settings, as explained in more detail below. Also, because the primal LP has more constraints than variables, the dual was solved instead. Since we are solving a finite LP, there is no duality gap and the dual solution is equal to the primal solution.

We also implemented a variant of the LP approach. In this variant, rather than use the state-action value during a *rest* phase, we perform dynamic programming (DP) on the current model estimate to determine a value function for the original MDP. The agent then chooses actions greedily based on this computed value function. This method is obviously much slower than the original LP approach presented in section 4.2, but it may produce better results.

[1] Note that we also used $\alpha = 0.1$ for the Q-learning portion of our algorithm.

4.4.1 Small MDP

For the small MDP problem we used the following parameter settings for the LP approach: $depth = 7$, $rest = 6$, and $maxSamples = 35$. Figure 4–4 (a) demonstrates the performance of the different algorithms. The y-axis is the average return per episode (*i.e.* total reward received divided by episode number).

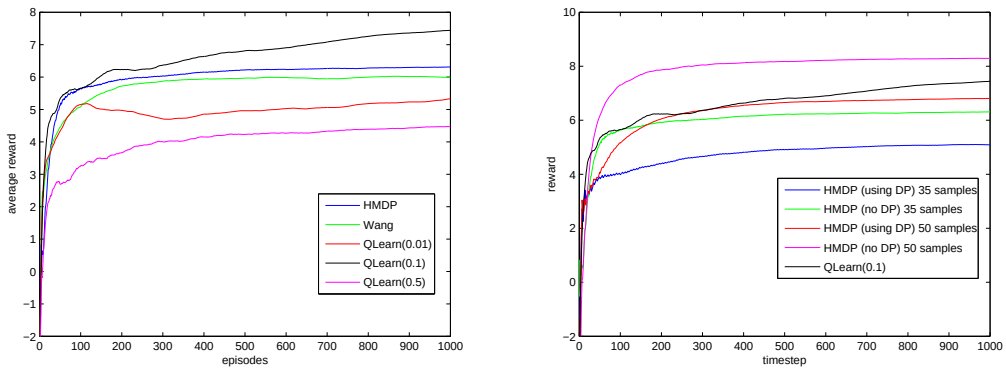

Figure 4–4: Small MDP: (a) Comparison of different algorithms; (b) Effect of varying parameters of the LP approach

The algorithm from (Wang et al., 2005) was not able to handle more than 35 samples, which is why we performed the comparison at this level. The LP approach was able to handle up to 50 samples, and the results of this are displayed in figure 4–4 (b). In the same figure we also display the results of running the LP approach with the DP variant mentioned above. Although Q-learning with $\epsilon = 0.1$ outperformed all the other algorithms in figure 4–4 (a), we can see in figure 4–4 (b) that if we allow the LP approach to use 50 samples for the sampled LP, it performs better than all the rest. Interestingly, using the DP variant produces worse results than the original LP approach. This is a very small problem so it

takes relatively few iterations before the action value estimates become accurate. This is probably occurring before our model estimate becomes accurate. Thus, using DP on this model estimate would produce worse results than using the action value estimates.

4.4.2 Bandit problem

For the bandit problem we used the following parameter settings for the LP approach: $depth = 6$, $rest = 5$, and $maxSamples = 45$, and the DP variant was used. Figure 4–5 (a) demonstrates the performance of the different algorithms. Once again, the y-axis is the average return per episode.

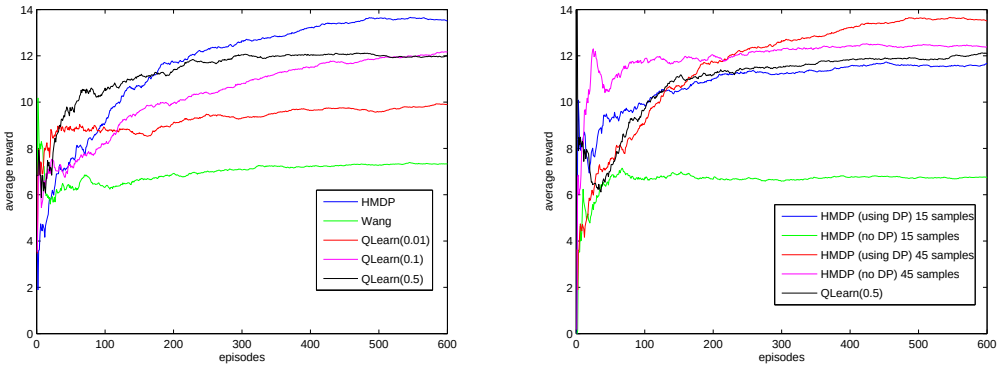

Figure 4–5: Bandit: (a) Comparison of different algorithms; (b) Effect of varying parameters of the LP approach

In this problem the algorithm from (Wang et al., 2005) was able to handle up to 45 samples. The LP approach outperformed all the other algorithms using 45 samples. In figure 4–5 (b) we plot the performance of the LP approach using fewer samples. We can see that even with only 15 samples the LP approach (with the DP variant) does almost as well as Q-learning with the best settings ($\epsilon = 0.5$)

and still performs better than the algorithm from (Wang et al., 2005). With fewer samples, however, not using the DP variant results in poor performance. This is a larger problem than the Small MDP problem above, so more iterations are necessary to obtain a good approximation of the action values. In this case, the model estimate is becoming more accurate faster than the action value estimates. Hence, using the DP variant produces improved performance than relying on action value estimates.

4.4.3 Grid world problem

For the grid world problem we used the following parameter settings for the LP approach: $depth = 9$, $rest = 8$, and $maxSamples = 15$, and we used the DP variant for the comparison. Figure 4–6 (a) demonstrates the performance of the different algorithms. This problem is different from the last two in that it *terminates*. When the agent reaches the goal state (the state with a reward of +1), it has *won* and is placed back in the start state. We allow all of the algorithms up to 60 time steps to reach the goal state. If by then the agent has not reached the goal state, the agent is placed back in the start state and a new episode begins. Whenever the agent reaches the goal state, it is also placed back in the start state and a new episode begins. The y-axis in figure 4–6 (a) is the sum of discounted rewards per episode. This means that the longer it took the agent to reach the goal, the less reward it received.

Once again, our choice of $maxSamples$ is due to the fact that the algorithm from (Wang et al., 2005) was not able to handle more than 15 samples. This is a larger problem than the first two, and all the other algorithms are 'stuck' with low

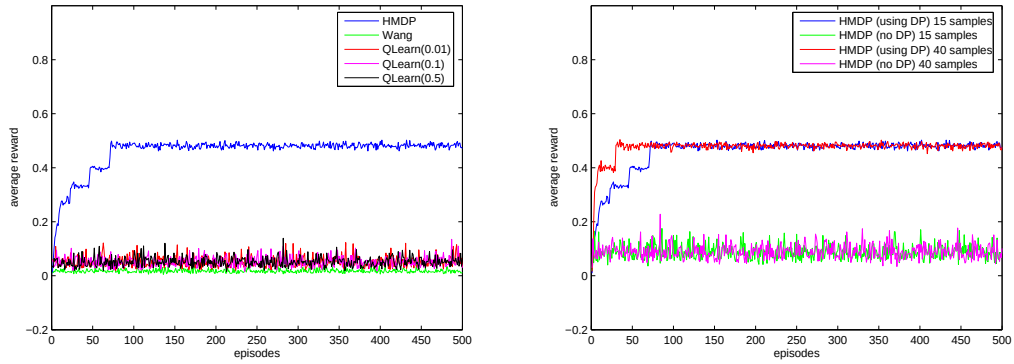

Figure 4–6: grid world: (a) Comparison of different algorithms on the grid world problem; (b) Effect of varying parameters of the LP approach

reward, while the LP approach is able to find a good policy. From figure 4–6 (b) it is clear that using the DP variant yields a clear advantage over the original LP approach. This problem is larger than both of the problems previously presented, so it is expected that it would take longer for the action value estimates to yield good results.

4.4.4 Running time

The last three sections demonstrate that the LP approach has an advantage over the other methods in terms of the reward obtained. However, in terms of computation time, the LP approach is obviously much slower than Q-learning or Boltzmann exploration. Table 4–1 plots the average running time per episode of the two variants of the LP approach along with the running time for the algorithm in (Wang et al., 2005). There is a notable difference between the two variants of the LP approach.

	HMDP		Wang
	Using DP	**No DP**	
Small MDP	0.0065	2.8408e-04	5.3850e-04
Bandit	0.0094	3.3234e-04	3.4113e-04
Grid world	0.0548	0.0083	0.0025

Table 4–1: Comparison of running times per episode (in seconds)

It should be noted that even when the DP variant was not used, in all cases the performance when using the LP approach was superior to the other algorithms.

4.5 Summary

In this chapter we have presented the LP approach to Bayesian exploration and showed that it compares favorably against other state of the art techniques for exploration. The LP approach is somewhat similar to the algorithm presented in (Wang et al., 2005). However, our approach exhibits better efficiency than the algorithm of (Wang et al., 2005).

Our algorithm is much slower in terms of running time than ϵ-greedy exploration, but in our experiments it achieves better performance. The algorithm in (Wang et al., 2005) has comparable running time but with worse performance.

CHAPTER 5
Near-optimal Bayesian learning

Although the LP approach enjoyed improved performance by sampling actions uniformly randomly, it is difficult to establish theoretical guarantees for it. In order to obtain formal guarantees we must expand all available actions at each sampled hyper state as required for the sparse sampling algorithm of (Kearns et al., 1999).

The problem domains used in chapter 4 are *toy* domains in the sense that they are useful for demonstrating characteristics of a particular algorithm or comparing it against others, but are nowhere near the size of a real world problem. Unfortunately, many state of the art algorithms which have formal guarantees do not scale well to large problems. In order to apply similar techniques, the designer must rely on heuristics or on embedding domain specific knowledge into the agent. Using heuristics can produce good results in practice, but the conditions required for the theoretical results are usually not present (Duff, 2002; de Farias and Roy, 2001). On the other hand, by embedding domain specific knowledge into the agent, the results may be very good for the desired problem, but fail to generalize to others.

In this chapter we present an algorithm that guarantees a near optimal exploration policy independent of the size of the state space and problem specification. Thus, we avoid the two pitfalls mentioned in the last paragraph.

As mentioned in section 2.4, many real world problems exhibit an underlying structure which can be exploited by representing the state space as a set of state variables, rather than explicitly listing all the states. We apply the algorithm presented in this chapter to a series of well known problems specified as factored MDPs and compare it against a state of the art exploratory algorithm for factored MDPs represented as ADDs.

5.1 The Algorithm

In chapter 4 we used the idea of sparse sampling from (Kearns et al., 1999), but we did not use the sparse sampling algorithm as presented in the paper in order to speed up computations. In this chapter we do make use of the sparse sampling algorithm as presented in (Kearns et al., 1999). The sparse sampling algorithm requires a generative black box to generate posterior states. As was mentioned in section 3.1.1, since our infinite Hyper MDP is built from a prior indexed by ψ, the model is fully known. This means that we have a generative model to obtain posterior states, namely, the distribution function $\mathbf{P}(\psi)$.

There are two main differences between the LP approach and the current approach. Rather than using a linear program to compute the value function for the sampled Hyper MDP, we use the approach from (Kearns et al., 1999), which is a form of value iteration (see section 2.2.1). Additionally, we expand all available actions at each hyper state, rather than sampling only a few as was done in the LP approach and in (Wang et al., 2005).

The full algorithm is described in Algorithm 3, with *SparseSample* described in Algorithm 4. Note that *SparseSample* has been modified slightly to include the generative model $\mathbf{p}(\psi)$.

Algorithm 3 bayesExplore($MDP, s_0, \psi, \epsilon, rest$)

$s \leftarrow s_0$
repeat
  Construct a model estimate from $H(\mathcal{P}|\psi)$
  $a \leftarrow SparseSample(\epsilon, \gamma, s, \psi)$
  Observe transition $(s \mapsto s')$ under action a
  $\psi \leftarrow D^a_{s,s'}(\psi)$
  $s = s'$
  if $rest > 0$ **then**
    Compute the value functions $\hat{V}(\cdot)$ based on the current model estimate
  end if
  for $i = 1$ to $rest$ **do**
    Choose action a greedily based on $\hat{V}(\cdot)$
    Observe transition $(s \mapsto s')$ under action a
    $\psi \leftarrow D^a_{s,s'}(\psi)$
    $s \leftarrow s'$
  end for
until done exploring

We make use of the *rest* parameter just as in the LP approach. In our current approach, we do something similar to the DP variant presented in section 4.4 in that at the beginning of each *rest* phase, we compute the value function based on the current model estimate. In contrast to the LP approach, we do not specify how to compute this value function. Indeed, Algorithm 3 does not specify how to maintain the model estimate nor how to compute the value functions $\hat{V}(\cdot)$. In other words, the algorithm is independent of the problem representation and the value function computation algorithm best suited for such a representation.

Algorithm 4 SparseSample$(\epsilon, \gamma, s, \psi)$

Define the horizon H and number of samples C as per Theorem 5.2.4
$\{Q^H(s,a,\psi)\}_{a\in\mathcal{A}} \leftarrow EstQ(H, C, \gamma, s, \psi)$
return $\arg\max_{a\in\mathcal{A}}\{Q^H(s,a,\psi)\}$

Procedure EstQ(h, C, γ, s, ψ)
if $h = 0$ **then**
  return $\{V_{max}, \cdots, V_{max}\}$
end if
for all $a \in \mathcal{A}$ **do**
  Generate C samples from $\mathbf{p}_s^a(\psi)$. Let S_a be the set containing all these C next states.
  $Q^H(s,a,\psi) \leftarrow \bar{r}_s^a(\psi) + \gamma\frac{1}{C}\sum_{s'\in S_a} EstV(h-1, C, \gamma, s', D_{s,s'}^a(\psi))$
end for
return $\{Q^H(s,a,\psi)\}_{a\in\mathcal{A}}$
end Procedure

Procedure EstV(h, C, γ, s, ψ)
$\{Q^H(s,a,\psi)\}_{a\in\mathcal{A}} \leftarrow EstQ(h, C, \gamma, s, \psi)$
return $\max_{a\in\mathcal{A}}\{Q^H(s,a,\psi)\}$
end Procedure

5.2 Theoretical analysis

As our algorithm is based on the sparse sampling algorithm of (Kearns et al., 1999), our theoretical analysis will also be based on the theoretical analysis of the same paper. The analysis presented in this section focuses on the computation of the value function for the Hyper MDP, $V(s,\psi)$, since this computation is what determines the exploration strategy of the agent.

Although the *rest* parameter increases the speed of the algorithm by reducing the number of times the Hyper MDP is constructed, it becomes problematic for our theoretical analysis. Thus, for this section, we will assume that *bayesExplore* (Algorithm 3) is used with $rest = 0$.

We begin by defining the estimator U^*:

$$U^*(s,a,\psi) = \bar{r}_s^a(\psi) + \gamma \frac{1}{C} \sum_{i=1}^{C} V^*(s_i, D_{s,s_i}^a(\psi))$$

where the s_i are drawn according to $\mathbf{p}_s^a(\psi)$. Note that $V^*(s,\psi)$ is the optimal value of hyper state (s,ψ). Although the optimal value function is inaccessible to us, defining the estimator U^* in this way is not problematic, as it is only necessary for the analysis.

From the definition of $V(s,\psi)$ in Equation 3.4, it follows that:

$$\begin{aligned} Q(s,a,\psi) &= \bar{r}_s^a + \gamma \sum_{s' \in \mathcal{S}} \bar{p}_{s,s'}^a(\psi) V(s', D_{s,s'}^a(\psi)) \\ &= \bar{r}_s^a + \gamma \mathbb{E}_{s' \sim \mathbf{p}_s^a(\psi)}[V^*(s', D_{s,s'}^a(\psi))]. \end{aligned}$$

In order to establish our main result (Theorem 5.2.4), we now present three short lemmas. The first and third of these are directly adapted from (Kearns et al., 1999), with minor modifications for the Hyper MDP framework. The second one is novel and is therefore included with proof.

Lemma 5.2.1 (Adapted from (Kearns et al., 1999)). *For any $s \in \mathcal{S}$, $a \in \mathcal{A}$ and $\psi \in \Psi$, with probability at least $1 - e^{-\lambda^2 C / {V_{max}}^2}$ we have:*

$$\begin{aligned} |Q^*(s,a,\psi) - U^*(s,a,\psi)| = \gamma \Big| & \mathbb{E}_{s' \sim \mathbf{p}_s^a(\psi)}[V^*(s', D_{s,s'}^a(\psi))] \\ & - \frac{1}{C} \sum_{i=1}^{C} V^*(s_i, D_{s,s_i}^a(\psi)) \Big| \leq \lambda \end{aligned}$$

where the probability is taken over the draw of the s_i from $\mathbf{p}_i^a(\psi)$.

Lemma 5.2.1 states that with "high" probability, the difference between the optimal value function V^* and the estimator U^* is "small". The qualitative

terms "high" and "small" are quantified by the choice of C (the number of samples) and λ. The required setting for these variables will be defined later on in Theorem 5.2.4.

For the following lemmas we will define $k = |\mathcal{A}|$. The next lemma will bound the combined error from truncating and sampling from the infinite hyper tree.

Lemma 5.2.2. *With probability at least* $1 - (kC)^n e^{-\lambda^2 C/{V_{max}}^2}$ *we have that*

$$|Q^*(s,a,\psi) - Q^n(s,a,\psi)| \leq \gamma\lambda + \gamma^n V_{max}$$

Proof.

$$\begin{aligned}
&|Q^*(s,a,\psi) - Q^n(s,a,\psi)| \\
&\qquad = \gamma \left| \mathbb{E}_{s' \sim \mathbf{p}_s^a}[V^*(s', D^a_{s,s'}(\psi))] - \frac{1}{C}\sum_{i=1}^{C} V^{n-1}(s_i, D^a_{s,s_i}(\psi)) \right| \\
&\qquad \leq \gamma \left(\left| \mathbb{E}_{s' \sim \mathbf{p}_s^a}[V^*(s', D^a_{s,s'}(\psi))] - \frac{1}{C}\sum_{i=1}^{C} V^{*}(s_i, D^a_{s,s_i}(\psi)) \right| \right. \\
&\qquad\quad \left. + \left| \frac{1}{C}\sum_{i=1}^{C} V^{*}(s_i, D^a_{s,s_i}(\psi)) - \frac{1}{C}\sum_{i=1}^{C} V^{n-1}(s_i, D^a_{s,s_i}(\psi)) \right| \right) \\
&\qquad = \gamma \left(\left| \mathbb{E}_{s' \sim \mathbf{p}_s^a}[V^*(s', D^a_{s,s'}(\psi))] - \frac{1}{C}\sum_{i=1}^{C} V^{*}(s_i, D^a_{s,s_i}(\psi)) \right| \right. \\
&\qquad\quad \left. + \left| \frac{1}{C}\sum_{i=1}^{C} \left[V^{*}(s_i, D^a_{s,s_i}(\psi)) - V^{n-1}(s_i, D^a_{s,s_i}(\psi)) \right] \right| \right) \\
&\qquad \leq \lambda + \gamma^n V_{max}
\end{aligned}$$

where the last inequality follows from Theorem 3.1.2 and lemma 5.2.1. At each level we require that each of the sampled next states be good estimates for each of the k actions. Lemma 5.2.1 showed that the probability of a bad estimate

is bounded by $e^{-\lambda^2 C/{V_{max}}^2}$. Thus the above bound holds with probability $1 - (kC)^n e^{-\lambda^2 C/{V_{max}}^2}$. □

Lemma 5.2.3 (Adapted from (Kearns et al., 1999)). *Assume that π is a stochastic policy, so that $\pi(s)$ is a random variable. If for each $s \in \mathcal{S}$ and $\psi \in \Psi$, the probability that $Q^*(s, \pi^*(s), \psi) - Q^*(s, \pi(s), \psi) < \lambda$ is at least $1 - \delta$, then the discounted infinite horizon return of π is at most $(\lambda + 2\delta V_{max})/(1 - \gamma)$ from the optimal return, i.e., for any $s \in \mathcal{S}$ and $\psi \in \Psi$, $V^*(s, \psi) - V^\pi(s, \psi) \leq (\lambda + 2\delta V_{max})/(1 - \gamma)$.*

Lemma 5.2.3 is a type of continuity result. It states that as the state-action value function induced by a particular policy π approaches the optimal state-action value function, the expected sum of the discounted rewards when using policy π approaches the optimal expected sum of discounted rewards.

We combine these three lemmas for our main result.

Theorem 5.2.4. *Algorithm 3 (with rest = 0) takes as input any state $s_0 \in \mathcal{S}$, any prior indexed by $\psi \in \Psi$, any value $\epsilon > 0$, and produces a policy π satisfying the following conditions:*

- *(Near-Optimality) The value function of the stochastic policy π produced by Algorithm 3 satisfies*

$$|V^\pi(s_0, \psi) - V^*(s_0, \psi)| \leq \epsilon.$$

- *(Efficiency) The approximation above is produced in time $O((kC)^H)$, where*

$$\begin{aligned} H &= \lceil \log_\gamma(\lambda/V_{max}) \rceil, \\ C &= \frac{V_{max}^2}{\lambda^2}\left(2H\log\frac{kHV_{max}^2}{\lambda^2} + \log\frac{V_{max}}{\lambda}\right), \\ \lambda &= \frac{\epsilon(1-\gamma)}{4}, \\ \delta &= \lambda/V_{max}. \end{aligned}$$

Proof. We begin by focusing on the near-optimality property. Lemma 5.2.2 showed the error of our Q-function approximations are bounded by $\lambda + \gamma^H V_{max}$ with probability $1 - (kC)^H e^{-\lambda^2 C/{V_{max}}^2}$. Setting $H = \log_\gamma(\lambda/V_{max})$ our final Q-value estimates $Q^H(s_0, a, \psi)$ for all $a \in \mathcal{A}$ when starting from hyper state (s_0, ψ) are within 2λ from the true Q-values. If we choose C as in (Kearns et al., 1999), namely

$$C = \frac{{V_{max}}^2}{\lambda^2}\left(2H\log\frac{k{H{V_{max}}^2}}{\lambda^2} + \log\frac{1}{\delta}\right)$$

then we guarantee that

$$(kC)^H e^{-\lambda^2 C/{V_{max}}^2} \leq \delta$$

thus ensuring that with probability $1 - \delta$ all the estimates are within 2λ. Lemma 5.2.3 implies that for each $s \in \mathcal{S}$ and $\psi \in \Psi$, the resulting policy π extracted from the Q^H functions yield

$$V^*(s, \psi) - V^\pi(s, \psi) \leq \frac{2\lambda + 2\delta V_{max}}{1-\gamma}$$

Substituting in $\delta = \lambda / V_{max}$ we obtain

$$V^*(s, \psi) - V^\pi(s, \psi) \leq \frac{4\lambda}{1 - \gamma}$$

Substituting in $\lambda = \epsilon(1 - \gamma)/4$ as specified we finally obtain

$$V^*(s, \psi) - V^\pi(s, \psi) \leq \epsilon$$

.

The efficiency of the algorithm follows in the same way as the proof of efficiency in (Kearns et al., 1999). **EstQ** calls **EstV** a total of kC times (C calls for each of the k actions). **EstV** then recursively calls **EstQ**, but reducing the horizon parameter h by 1. Thus, the depth of the recursion is at most H, the desired horizon. It then follows that the running time is $O((kC)^H)$. □

Theorem 5.2.4 demonstrates that the value of a hyper state (s, ψ) can be approximated efficiently to within a desired ϵ, independent of the size of the state space. Note that, unlike other PAC results, the confidence parameter δ is not specified beforehand, but is dependent on the chosen accuracy parameter, ϵ. To the best of our knowledge, there is no previous algorithm for performing near-optimal Bayesian learning in factored MDPs such that computational complexity is independent of domain size.

5.3 Empirical analysis

We ran *bayesExplore* on various experimental domains to observe its performance in practice. We compared the results against the LP approach and against R-MAX (see section 2.5.4). As mentioned in the start of this chapter, our problem

domains are all standard problems used to test algorithms for factored MDPs. In the case of the LP approach, the domain was "flattened out" from a factored representation into a standard MDP representation. For all domains the results were averaged out over 10 independent runs.

5.3.1 Experimental results

As mentioned at the start of the chapter, we will consider factored MDPs represented as ADDs and use Algorithm 3 to perform exploration. We use SPUDD (see section 2.4.1) to compute the value function $\hat{V}(\cdot)$ in algorithm 3 at the start of each *rest* phase.

For our experimental results we use a standard domain for factored MDPs, namely, the SysAdmin problem introduced in (Guestrin et al., 2001). The problem consists of a network of computers linked together with some particular topology. Each computer in the network can be in one of two possible states: *up* or *down*. The agent receives a reward of 1 for each computer that is up, and the goal for the system administrator is to keep as many of the computers up as possible. Each computer has some probability of going down at each time step, and this probability is increased as the number of neighbors that are down increases. There is a hierarchical order in the connectivity of computers in that the state of 'parent' computers can influence the state of the child computers, but not the other way. In the framework presented in (Guestrin et al., 2001), at each time step the system administrator has two actions choices: reboot one of the machines or do nothing. When a machine is rebooted it deterministically goes up.

We ran experiments on the problem domain just mentioned and on a slight modification of it. The modification is to add a new state for each machine and a new action. The new action is to reboot a computer, which sends it into a *rebooting* state with high probability, and with small probability it could make the computer go up or down. The agent receives a penalty of -1 for each computer that is in a *rebooting* state[1] . This added action and state are meant to try to *trick* the agent into choosing a bad action. They serve to demonstrate how the far-sightedness of *bayesExplore* prevents the agent from choosing bad actions.

The first network topology we considered is the simplest one, which is that all the computers are connected in a ring (see Figure 5–1). The state of each computer is only affected by the state of its immediate neighbors. We shall refer to this problem as the Ring$n$ problem in its original formulation, where n is the number of computers on the network, and as the Ring$n$' problem for the modification with the added action.

We first present the results comparing the *bayesExplore* algorithm against the LP approach on the Ring4 problem. Figure 5–2 (a) plots average reward versus iteration number and Figure 5–2 (b) plots cumulative reward versus the log of the time. These results clearly demonstrate that *bayesExplore* yields higher returns and is faster than the LP approach. Unfortunately, the Ring4 problem was the

[1] Note that because of assumption 2.1.1 we shift the rewards up so they are positive

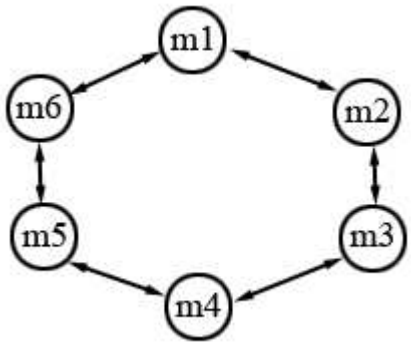

Figure 5–1: Ring Topology with 6 computers

largest problem of the SysAdmin problems that the LP approach was able to handle.

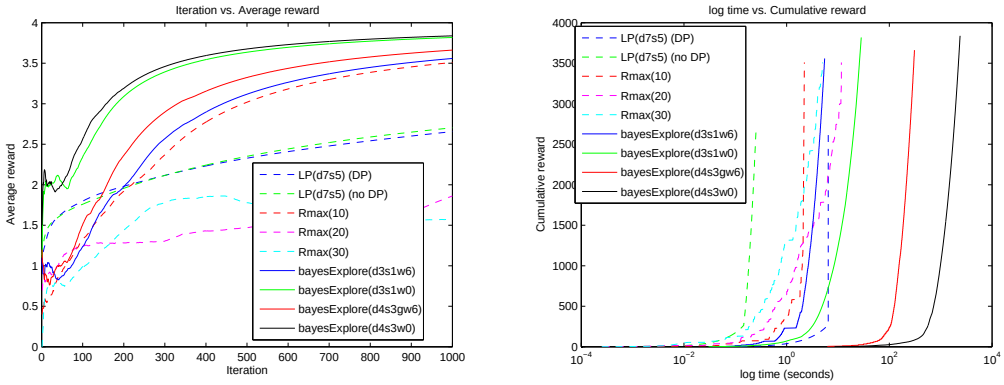

Figure 5–2: Ring4 problem: (a) Average reward versus iteration number; (b) Cumulative reward versus log time

Now we compare *bayesExplore* against R-MAX on the Ring4' problem. As mentioned previously, the LP approach was not able to handle this problem. The results are presented in Figure 5–3. In Figure 5–3 (a), where we plot average reward versus iteration number, the superior performance of *bayesExplore* is evident. Needless to say, *bayesExplore* is much slower than R-MAX. For this

reason, we allowed R-MAX to run for more iterations in order to compare their performance in terms of elapsed time, rather than iteration number. Figure 5–3 (b) presents these results, and they clearly demonstrate that *bayesExplore* obtains a much higher return even when we allow R-MAX to run for the same amount of time.

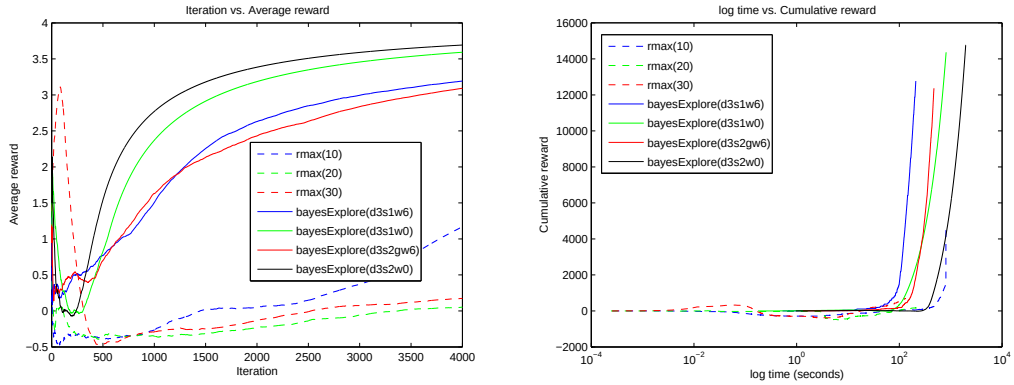

Figure 5–3: Ring4' problem: (a) Average reward versus iteration number; (b) Cumulative reward versus log time

The second network topology we considered is where there is a central server and the network machines branch out from this central server (see Figure 5–4). In this problem, rather than rebooting (or reinstalling) each machine individually, the system administrator can *emit* a reboot or reinstall action through a particular leg. The effect of the action decays by a factor β the further down the leg the machine is located. This means that if a reboot action is issued to a leg, the first machine on the leg will come up deterministically, the second machine on the leg will come up with probability β and will stay in its current state otherwise, the third machine on the leg will come up with probability β^2, etc. The effect of the

reinstall action is decayed by the same amount β. The probability of a machine going down is increased if its parent (the machine directly above it on the leg) is down. Note that this means that the machines closest to the server are not affected by the state of any other machines. Figure 5–4 illustrates these dynamics. We will refer to these problems as Legs_m^n, where n is the number of legs and m is the number of machines per leg.

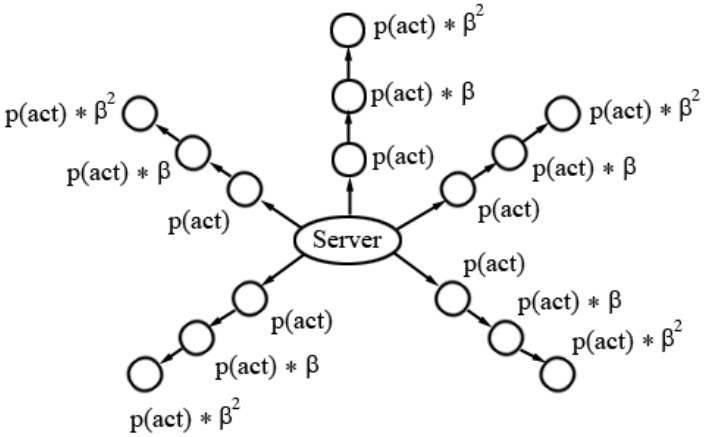

Figure 5–4: Legs topology with 5 legs and 3 computers per leg and decaying dynamics for action *act*

We compared *bayesExplore* against R-MAX using various settings for n and m. The results for the Legs_2^2 problem are presented in Figure 5–5; the results for the Legs_3^2 problem are presented in Figure 5–6; and the results for the Legs_2^3 problem are presented in Figure 5–7. These problem domains are larger than the SysAdmin' problem domains, so the performance of all algorithms is worse. Nevertheless, *bayesExplore* clearly has an advantage over R-MAX and the results demonstrate that an increased performance is achieved with a greater depth and

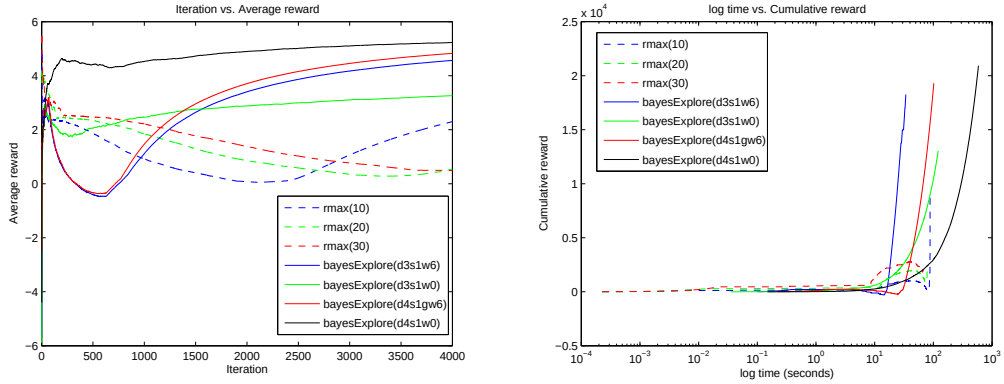

Figure 5–5: Legs$_2^2$ problem: (a) Average reward versus iteration number; (b) Cumulative reward versus log time

number of samples. We can also observe that although the *rest* parameter helps speed up the algorithm, it deteriorates its performance.

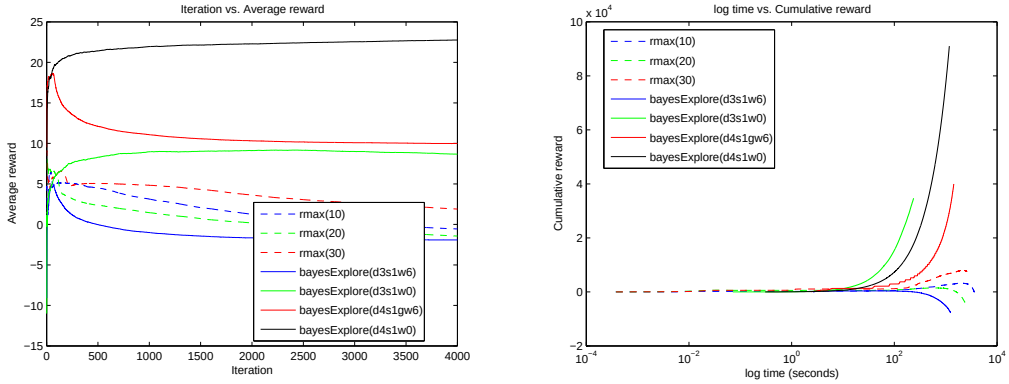

Figure 5–6: Legs$_3^2$ problem: (a) Average reward versus iteration number; (b) Cumulative reward versus log time

5.4 Summary

In this chapter we demonstrated that we can approximate the value of a hyper state (s, ψ) up to a desired level of accuracy dependent only on R_{max}, γ and ϵ,

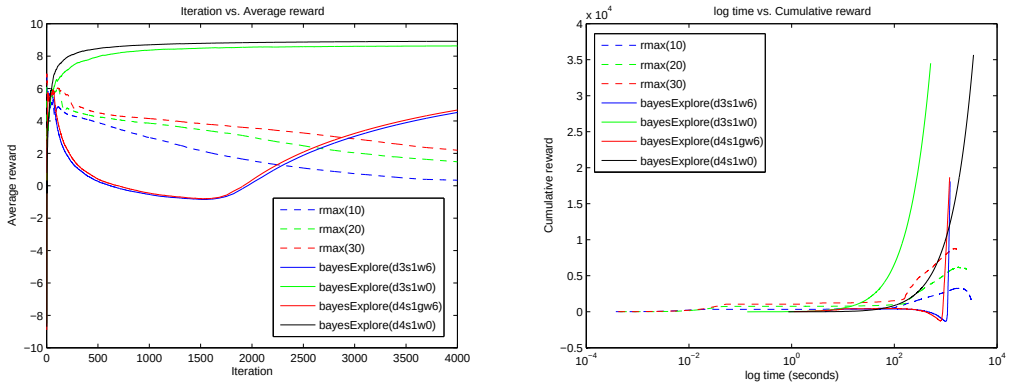

Figure 5–7: Legs$_2^3$ problem: (a) Average reward versus iteration number; (b) Cumulative reward versus log time

and independent of the size of the state space. This is an enormous advantage for exploration, as it allows us to deal with a much larger range of problems.

Although methods that use Approximate Linear Programming (ALP) (de Farias and Roy, 2003; Hauskrecht and Kveton, 2004; Guestrin et al., 2002) also enjoy this benefit, the choice of basis functions is not immediately clear and is specific to each domain. Furthermore, ALP methods reduce the number of variables in the original LP to a manageable size, but the number of constraints may still remain arbitrarily large. de Farias and Roy (2001) present a method for sampling a subset of the constraints. Their results, however, depend on knowledge of the optimal policy, which is not known beforehand.

The *bayesExplore* algorithm also enjoys an independence from the problem specification, which allows it to be applicable for a large range of problems of varying sizes and structures. For our experiments we still chose small problems to simplify the comparison of the various algorithms.

It is often the fact that algorithms which enjoy rich theoretical guarantees tend to not perform as well in practice. To demonstrate the strong empirical performance of *bayesExplore*, we compared it against other state of the art exploratory techniques (the LP approach of chapter 4 and (Brafman and Tennenholtz, 2001)). Our results indicate that *bayesExplore* has a strong empirical performance, even when the depth and number of samples used is very small.

CHAPTER 6
Projections in infinite dimensional spaces

After having computed the value function for a sampled hyper MDP rooted at hyper state (s, ψ) and performing an action leading to hyper state (s', ψ'), in the previous algorithms we are obliged to recompute the value function for the hyper MDP rooted at (s', ψ') from scratch. Nevertheless, there may be a substantial amount of overlap between the two hyper trees. One would hope that the value function computed in one iteration can be used as a starting point for the computation of the value function in the next iteration. This idea motivated the algorithm presented in this chapter, based on the Simplex algorithm for LPs. The Simplex algorithm is the most widely known algorithm for solving a finite LP, where at each iteration the algorithm goes from one extreme point to another until the optimum is reached (Chvátal, 1983). In order to achieve this, we must deal with finite *projections* of our original problem.

As was mentioned in section 4.1, the LP formulation of the optimality equations for hyper MDPs defined by equations (4.1) is a countably infinite dimensional LP. There has been a growing body of work on infinite dimensional LPs recently, mainly spurred by the seminal book of Anderson and Nash (1987). A convenient property of finite LPs is that there is no duality gap. This means that the optimal value of the primal and dual LPs agree when a solution exists (Chvátal, 1983). Unfortunately, in infinite dimensional spaces this is not always

the case, and there are many simple infinite dimensional LPs that suffer from a duality gap. In (Anderson and Nash, 1987) some conditions that guarantee absence of a duality gap are presented, which rely on the topological properties of the spaces in question.

The theory of infinite dimensional LPs has recently been applied with some success to MDPs for the average reward case (Hernández-Lerna and Lasserre, 2002; Hernández-Lerma and Lasserre, 1996; Lasserre, 1992; Hordijk, 1994; Klabjan and Adelman, 2006). However, there has been very little work on the discounted reward criterion. In (Hernández-Lerna, 1989) the Bayesian model explored in this thesis is discussed, but the algorithmic contributions are similar to what was already presented in (Martin, 1967). An interesting method was presented in (Ghate et al., 2007) with an application to the discounted reward criterion. The method presented in this chapter is based on the idea in that paper, but our problem formulation and theoretical results are significantly different.

In this chapter we present an anytime algorithm for Bayesian exploration. An anytime algorithm is one that will continue to refine and improve its solution until its time "runs out". Once its time runs out, it returns the latest solution, and the more time it has, the better the solution returned will be. In fact, we will present the algorithm and its theoretical properties for the general class of MDPs with countable state spaces, finite actions and bounded rewards, of which Hyper MDPs are a special case. We begin by providing some background on infinite dimensional LPs, followed by the algorithm, its theoretical properties and experimental results.

6.1 Background

Infinite Dimensional spaces are extremely interesting in their own right and have received a great deal of attention. They are widely used in applied areas such as Economics, Control Theory and Operations Research. They are generally more difficult to work with than finite spaces, as many properties which are taken for granted in finite dimensional spaces fail to hold in infinite dimensional spaces, unless certain specialized conditions hold. For the most part, we will focus on the theory of Linear Programming in infinite dimensional spaces in this section. We take for granted that the reader has some basic knowledge of topology. If not, (Munkres, 2000) provides a great introduction.

As mentioned in the introduction to this chapter, the absence of a duality gap in finite LPs is a property that fails to hold in infinite dimensional LPs. Furthermore, the continuity of the objective function and of the constraints are not guaranteed in arbitrary spaces. In order to show that these properties hold in infinite dimensional spaces, we must first formulate the LP problem in a topological setting. Most of the notation used for this presentation is taken from (Aliprantis and Border, 2006) and (Anderson and Nash, 1987).

Definition 6.1.1. *A* **topological vector space** *(tvs) X is a vector space over $\mathbb{R}$ with a translation-invariant topology having a neighborhood base $\mathcal{B}$ at zero with the following properties:*

1. *Each $V \in \mathcal{B}$ is* **absorbing**. *That is, for each vector $x \in X$ there exists some $\alpha_0 > 0$ such that $\alpha x \in V$ whenever $-\alpha_0 \leq \alpha \leq \alpha_0$.*

2. *Each $V \in \mathcal{B}$ is* **circled**. *That is, for each $v \in V$ and any $|\alpha| \leq 1$ we have $\alpha v \in V$.*

3. *For each $V \in \mathcal{B}$ there exists some $W \in \mathcal{B}$ with $W + W \subset V$.*

4. *Each $V \in \mathcal{B}$ is closed.*

Having defined the topological space on which most of our work will be done, we will now define dual pairs, which are essential for a proper formulation of a general LP.

Definition 6.1.2. *Given a topological vector space X, its* **topological dual** X' *is the vector space of all continuous linear functionals on X.* [1]

We can now define dual pairs, but before doing so, we must define a type of functional on the product of two spaces.

Definition 6.1.3. *A* **bilinear functional** *for two tvs X and Y is some function from $X \times Y$ to $\mathbb{R}$, written as $\langle \cdot, \cdot \rangle$, with $\langle x, y \rangle$ a linear function of x for each fixed $y \in Y$, and a linear function of y for each fixed $x \in X$.*

Definition 6.1.4. *A* **dual pair** *is a pair (X, X') of vector spaces together with a bilinear functional $(x, x') \mapsto \langle x, x' \rangle$, from $X \times X'$ to $\mathbb{R}$, that separates the points of X and X'. That is:*

1. *$\langle \cdot, \cdot \rangle$ is a bilinear functional on $X \times X'$.*

2. *The mapping $x \mapsto \langle x, x' \rangle$ is linear for each $x' \in X'$.*

3. *If $\langle x, x' \rangle = 0$ for each $x' \in X'$, then $x = 0$.*

[1] This is a strict subset of the **algebraic dual** X^*, the vector space of all linear functionals on X.

4. If $\langle x, x' \rangle = 0$ *for each* $x \in X$*, then* $x' = 0$*.*

Many necessary properties required for our optimization problem require the space to be convex (*i.e.* including the line segment joining any two of its points). A Hausdorff tvs having a neighborhood base at zero of convex sets is called a **locally convex space**. In order to guarantee that we are dealing with a pair of locally convex spaces (X, X'), the most commonly topologies used on X and X' are the weak and weak* topologies, defined below.

Definition 6.1.5. *Given two spaces* X *and* X'*, the* **weak topology** *on* X *(denoted* $\sigma(X, X')$*) is the topology having a base of neighborhoods at the origin composed of sets of the form* $B_A = \{x \in X : -1 \leq \langle x, x' \rangle \leq 1. \qquad \forall x' \in A\}$*, where* A *runs through all the finite subsets of* X'*.*

The **weak* topology** *on* X' *(denoted* $\sigma(X', X)$*) is defined in a similar manner.*

If (X, X') is a dual pair, then the topological dual of the tvs $(X, \sigma(X, X'))$ is X' (Aliprantis and Border, 2006).

Remark 6.1.6. *There may be many topologies which are* consistent *with the dual pair* (X, X')*. Indeed, every locally convex Hausdorff space* (X, τ) *is consistent with the dual pair* (X, X') *(see (Aliprantis and Border, 2006)).*

For formulating an LP, we need to look at two dual pairs of tvs, let (X, Y) and (Z, W) be such pairs, equipped with their respective weak topologies $\sigma(X, Y)$ and $\sigma(Z, W)$. Let A be a linear map from X to Z, its *adjoint* (or transpose) A^* is defined by the relation $\langle Ax, w \rangle = \langle x, A^* w \rangle$ for each $x \in X$ and $w \in W$. A^* is a linear map from W to the X^* (the algebraic dual of X). However, this may be

problematic since, as mentioned before, Y is a strict subset of X^*. The following proposition from (Anderson and Nash, 1987) remedies this situation.

Proposition 6.1.7. *A^* maps W into Y if and only if A is continuous with respect to the topologies $\sigma(X,Y)$ and $\sigma(Z,W)$.*

We are now ready to formulate the Linear Programming problem in a topological setting. Heretofore we will always assume that we are using a topology consistent with the dual pair in question. We denote the null vector by θ. Let $\mathfrak{P}$ and $\mathfrak{Q}$ be the positive convex cones in X and Z respectively (a convex cone is a set closed under addition and multiplication by positive scalars), and let A be a $\sigma(X,Y) - \sigma(Z,W)$-continuous map from X to Z. An inequality constrained LP can then be defined as follows:

$$\begin{aligned} \text{IP: minimize} \quad & \langle x, c \rangle \\ \text{such that} \quad & Ax - b \geq \theta, \\ & x \geq \theta \end{aligned} \tag{6.1}$$

where $b \in Z$ and $c \in Y$. As in (Anderson and Nash, 1987), we will refer to the above linear program as IP.

Before formulating the dual LP, we must define the *dual cones* of $\mathfrak{P}$ and $\mathfrak{Q}$:

$$\mathfrak{P}^* = \{y \in Y : \quad \langle x, y \rangle \geq 0 \quad \forall x \in \mathfrak{P}\}$$
$$\mathfrak{Q}^* = \{w \in W : \quad \langle z, w \rangle \geq 0 \quad \forall z \in \mathfrak{Q}\}$$

The dual LP (which we will refer to as IP*) can now be defined as follows:

$$\begin{aligned} \text{IP}^*\text{: maximize} \quad & \langle b, w \rangle \\ \text{such that} \quad & -A^*w + c \in \mathfrak{P}^* \\ & w \in \mathfrak{Q}^* \end{aligned} \tag{6.2}$$

Note that the above formulations are a generalized form of representing LPs, and the LPs presented previously fall within this category. We will refer to X as the primal variable space, Z as the primal constraint space, W as the dual variable space and Y as the dual constraint space.

Although strong duality does not always hold for infinite dimensional programs, we do have weak duality, as stated in the following theorem from (Anderson and Nash, 1987).

Theorem 6.1.8. *If IP and IP* are both consistent (*i.e. *have a feasible solution), the the value of IP is greater than or equal to the value of IP* and both values are finite.*

Another subtlety which is taken for granted in finite dimensional LPs is that the dual of the dual (IP**) is exactly equal to IP. In infinite dimensional cases this is only the case if the positive cone $\mathfrak{P}$ is closed.

We close this section by presenting a theorem from (Anderson and Nash, 1987) which specifies a sufficient condition for the absence of a duality gap. Before doing so, we need to define a special set D in $Z \times \mathbb{R}$ as follows:

$$D = \{(Ax - z, \langle x, c \rangle) : x \in \mathfrak{P}, z \in \mathfrak{Q}\}.$$

Theorem 6.1.9. *If IP is consistent, with finite value, and D is closed, then there is no duality gap for IP.*

6.2 Absence of a duality gap

We will be dealing with subsets of the sequence spaces ℓ_∞ (bounded sequences) and ℓ_1 (absolutely summable sequences). To be more precise, $\ell_\infty = \{x \in \mathbb{R}^\infty : \sup |x_i| < \infty\}$ and $\ell_1 = \{x \in \mathbb{R}^\infty : \sum_{i=1}^{\infty} |x_i| < \infty\}$.

Continuing with the notation in section 6.1, we will use a subspace $\mathfrak{V} \subset \ell_\infty$ as our primal variable space X, and a subspace $\mathfrak{R} \subset \ell_1$. Let $\mathfrak{P}$ and $\mathfrak{Q}$ denote the positive cones of ℓ_∞ and ℓ_1, respectively. The subspaces are defined as follows:

$$\mathfrak{V} = \{x : x \in \ell_\infty, 0 \leq x_i \leq V_{max} \forall i \in \mathbb{N}\}$$

$$\mathfrak{R} = \{z : z \in \ell_\infty, 0 \leq z_i \leq R_{max} \forall i \in \mathbb{N}\}$$

From the above definitions we clearly have $\mathfrak{V} \subset \mathfrak{P}$ and $\mathfrak{R} \subset \mathfrak{Q}$.

As mentioned in the introduction to this chapter, we will first present our results for a general class of MDPs, and then show that Hyper MDPs are in fact a part of this class and so all the results proved will immediately apply. Consider an MDP as defined in section 2.1 with a countably infinite number of states, a finite number of available actions and respecting assumption 2.1.1.

Consider the primal LP formulation presented in section 2.2.3, rephrased for our current context.

$$\begin{aligned} \text{minimize} \quad & \langle x, c \rangle \\ \text{such that} \quad & Ax \geq b, \\ & x \geq \theta \end{aligned} \tag{6.3}$$

In LP (6.3), $x \in \ell_\infty$ denotes the vector containing the state values for all the states in the MDP (*i.e.* x_i holds the value of state $i \in \mathcal{S}$) and c denotes the vector containing the state-relevance weights (*i.e.* c_i holds the state relevance weight for state i). We require that $c \in \ell_1$ for the following reason. As previously mentioned, for infinite dimensional LPs we must ensure that we are dealing with topological dual pairs. The dual of ℓ_∞ is $\ell_1 \oplus \ell_1^d$, where ℓ_1^d is the disjoint complement of ℓ_1 and consists of all singular functionals (see (Aliprantis and Border, 2006)). By restricting c to an element of ℓ_1, we are guaranteeing that the requirement of topological dual pairs is respected, thus ensuring the continuity of the objective function and constraints. One possibility for the choice of c is that of a probability distribution over the state space. Such a distribution could represent the starting state distribution of the MDP. For Hyper MDPs the choice of c could be that of a probability distribution defined over the infinite hyper tree assigning higher probabilities the closer to the root a hyper state is.

The linear transform A is indexed by $[(i,a),j]$, where $i,j \in \mathcal{S}$ and $a \in \mathcal{A}$. Thus, $A := [a_{(i,a),j}]$ where $A_{(i,a)}$ is a vector defined by

$$A_{(i,a)} = [a_{(i,a),j}] = \begin{cases} 1 - \gamma P(i,a,i) & \text{if } i = j \\ -\gamma P(i,a,j) & \text{otherwise} \end{cases}.$$

$b \in \ell_\infty$ is defined by $b := [b_{(i,a)}] = [\mathcal{R}(s,a)]$. Note that the product $A_{(i,a)} \cdot x$ is bounded, as demonstrated by the following derivation:

$$\begin{aligned} A_{(i,a)} \cdot x &= x_i - \sum_{j \in \mathcal{S}} \gamma P(i,a,j) x_j \\ &\leq V_{max} - \sum j \in \mathcal{S} \gamma P(i,a,j) V_{min} \\ &= V_{max} - \gamma V_{min} \sum_{j \in \mathcal{S}} P(i,a,j) \\ &= V_{max} - \gamma V_{min} \end{aligned}$$

since P is a probability distribution. Thus, the range of A is ℓ_∞ as required.

Let F be the feasible region for LP (6.3) (*i.e.* $F = \{x : x \in \ell_\infty, Ax \geq b, x \geq \theta\}$). From assumption 2.1.1 it is clear that $x \in \mathfrak{V}$ whenever $x \in F$ and furthermore, $b \in \mathfrak{R}$.

With this in mind, we will define $D' \subset D$ as follows:

$$D' = \{(Ax - z, \langle x, c \rangle) : x \in \mathfrak{V}, z \in \mathfrak{R}\}.$$

Lemma 6.2.1. *D' is closed.*

Proof. Let $(\xi^{(\alpha)}, r^{(\alpha)}) \to (\xi, r)$ in $Z \times \mathbb{R}$ with $(\xi^{(\alpha)}, r^{(\alpha)}) \in D'$ for all α. Each $(\xi^{(\alpha)}, r^{(\alpha)})$ is of the form $(Ax^{(\alpha)} - z^{(\alpha)}, \langle x^{(\alpha)}, c \rangle)$ where each $x^{(\alpha)} \in \mathfrak{V}$ and each

$z^{(\alpha)} \in \mathfrak{R}$. This means $(Ax^{(\alpha)} - z^{(\alpha)}) \to (Ax - z)$ and $\langle x^{(\alpha)}, c\rangle \to \langle x, c\rangle$. We need to show $x \in \mathfrak{V}$ and $z \in \mathfrak{R}$.

Let $x = \lim_{\alpha\to\infty} x^{(\alpha)}$. Since $x^{(\alpha)} \leq (V_{max})$ (the constant V_{max} sequence) for all α it requires only a moment's thought to see that $x \leq (V_{max})$, implying $x \in \mathfrak{V}$.

Now let $z = \lim_{\alpha\to\infty} z^{(\alpha)}$. Since $z^{(i)} \leq (R_{max})$ (the constant R_{max} sequence) for all α it is also clear that $z \leq (R_{max})$, implying $z \in \mathfrak{R}$. Thus, $(Ax - z, \langle x, c\rangle) \in D'$, completing the proof. □

An immediate corollary of this result is the following:

Corollary 6.2.2. $\mathfrak{V}$ *is closed.*

We may now prove the following important theorem:

Theorem 6.2.3. *There is no duality gap for LP (6.3).*

Proof. Theorem 2.2.1 asserts that an optimal solution to LP (6.3) exists, which confirms the LP is consistent. Lemma 6.2.1 demonstrated that D' is closed. Since all the conditions for theorem 6.1.9 are met, the result follows. □

Furthermore, corollary 6.2.2 implies the dual of the dual of LP (6.3) is equal to LP (6.3) itself.

We have thus shown that for the general class of MDPs described above, the solutions to the primal and dual infinite dimensional LPs agree. To the best of the author's knowledge, this is the first time this result has been proven, as finiteness of the state space has always been assumed when using LPs to solve the Bellman optimality equations (Puterman, 1994),(Bertsekas and Tsitsiklis, 1996). In the

next section we present a characterization of the extreme points of these infinite dimensional LPs and a method for approximating them through finite LPs.

6.3 Convergent projections

In (Cross et al., 1998) the authors characterize the set of extreme points of an infinite dimensional compact convex set and their finite projections. Before presenting their results, we need a few more definitions.

Definition 6.3.1. *A function $f : C \mapsto \mathbb{R}$ on a convex set C in a vector space is* **convex** *if $f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y)$ for all $x, y \in C$ and all $0 \leq \alpha \leq 1$; it is* **concave** *if $-f$ is a convex function.*

Note that a linear function is both convex and concave.

Definition 6.3.2. *A function $f : X \mapsto [-\infty, \infty]$ on a topological space X is:*

- **lower semi-continuous** *if for each $c \in \mathbb{R}$ the set $\{x \in X : f(x) \leq c\}$ is closed.*
- **upper semi-continuous** *if for each $c \in \mathbb{R}$ the set $\{x \in X : f(x) \geq c\}$ is closed.*

Note that a real function is continuous if and only if it is both upper and lower semi-continuous.

Definition 6.3.3. *A point $x \in S$ is called an extreme point of S if x is not the midpoint of any line segment contained in S.*

By the Bauer Minimum Principle (see (Aliprantis and Border, 2006)), when the feasible region of an optimization problem is a nonempty compact convex

subset $S \subseteq \mathbb{R}^\infty$, and the minimizing objective function is a concave lower semi-continuous function on S, then the optimum is attained at an extreme point of S.

Now we will define the projection function presented in (Cross et al., 1998).

Definition 6.3.4. *For each $N = 1, 2, \cdots$, define the projection function $p_N : \mathbb{R}^\infty \mapsto \mathbb{R}^N$ as $p_N(x) = (x_1, \cdots, x_n)$ and the corresponding projections of S onto $\mathbb{R}^N$ as $S_N = \{p_N(x) : x \in S\}$.*

It will sometimes be necessary to view S_N as a set embedded in $\mathbb{R}^\infty$ via $S_N = \{(p_N(x), \mathbf{0}) : x \in S\}$. Let E be the set of extreme points of S and E_N be the set of extreme points of S_N, where E_N can also be thought of as a set embedded in $\mathbb{R}^\infty$.

The following is the main result from (Cross et al., 1998).

Theorem 6.3.5. *The sets of extreme points E_N of the projections S_N of the compact convex set S converge to the closure of the extreme points of S, i.e. $\lim_{N\to\infty} E_N = \bar{E}$. If E is closed then we have $\lim_{N\to\infty} E_N = E$.*

We highlight that the authors assume that $\mathbb{R}^\infty$ is a space equipped with the product topology. In our problem, we are dealing with elements of the space $\mathfrak{V} \subset \ell_\infty \subset \mathbb{R}^\infty$, and thus, the results from (Cross et al., 1998) apply immediately to our situation. Furthermore, lemma 6.3.6 below from (Aliprantis and Border, 2006) combined with remark 6.1.6 ensure that the product topology on ℓ_∞ is consistent with the dual pair in question.

Lemma 6.3.6. *The product topology on $\mathbb{R}^\infty$ is a complete locally convex Hausdorff topology.*

We proceed to prove some necessary lemmas, recalling that F is the feasible region of LP (6.3).

Lemma 6.3.7. *F is compact.*

Proof. A point $x \in F$ is of the form $(x_1, x_2, \cdots, x_i, \cdots)$, where each $i \in \mathcal{S}$. From assumption 2.1.1 we have that $x_i \in [0, V_{max}]$ for all i. Since $[0, V_{max}]$ is compact, by the Tychonoff product theorem (see (Aliprantis and Border, 2006)) it follows that F is compact. □

Lemma 6.3.8. *F is convex.*

Proof. Let x, y bet two points in F. We know there is at least one such point by theorem 2.2.1, and if there is only one point then the lemma is vacuously true. Consider $z = \lambda x + (1-\lambda)y$ for some $0 \leq \lambda \leq 1$. x is of the form $(x_1, x_2, \cdots, x_i, \cdots)$ and y is of the same form. z is thus of the form $(\lambda x_1 + (1-\lambda)y_1, \cdots, \lambda x_i + (1-\lambda)y_i, \cdots)$. We need to show that the constraints of LP (6.3) are satisfied by z.

$\lambda x_i + (1-\lambda)y_i$ is clearly positive since x_i,y_i,λ and $(1-\lambda)$ are all positive. Now take any action $a \in \mathcal{A}$, we have that

$$\begin{aligned}
z_i - \gamma \sum_{j \in \mathcal{S}} P(i,a,j) z_j &= \lambda x_i + (1-\lambda)y_i - \gamma \sum_{j \in \mathcal{S}} P(i,a,j)(\lambda x_j + (1-\lambda)y_j) \\
&= \lambda x_i - \gamma \sum_{j \in \mathcal{S}} P(i,a,j)\lambda x_j + (1-\lambda)y_i - \gamma \sum_{j \in \mathcal{S}} P(i,a,j)(1-\lambda)y_j \\
&= \lambda \left(x_i - \gamma \sum_{j \in \mathcal{S}} P(i,a,j) x_j \right) + (1-\lambda) \left(y_i - \gamma \sum_{j \in \mathcal{S}} P(i,a,j) y_j \right) \\
&\geq \lambda \mathcal{R}(i,a) + (1-\lambda)\mathcal{R}(i,a) \\
&= \mathcal{R}(i,a)
\end{aligned}$$

where the inequality follows from the fact that x and y are both feasible.

Since a was arbitrary, we have that $z \in F$. Since λ was arbitrary, it follows that F is convex. □

Since our objective function $\langle x, c \rangle$ is linear, it follows that it is both convex and concave. Further, since $x \in \ell_\infty$, $c \in \ell_1$ and the dual of ℓ_∞ is $\ell_1 \oplus \ell_1^d$, we have that $\langle x, c \rangle$ is continuous.

We have thus demonstrated that all the required conditions for theorem 6.3.5 are met. Ordering the variables for each hyper state in some fashion, define the projections F_N of F in a similar way as the projections S_N were defined, namely $F_N = \{p_N(x), x \in F\}$. However, rather than appending a vector of zeros when embedding the projection in $\mathbb{R}^\infty$, we will append a vector of V_{max} values. In other words, we can consider F_N as a set embedded in $\mathbb{R}^\infty$ via $F_N = \{(p_N(x), \mathbf{V_{max}}) : x \in F\}$. Let E represent the set of extreme points of F and E_N the set of extreme points of the projection F_N, for all N. The following result follows immediately from theorem 6.3.5.

Theorem 6.3.9. *The sets of extreme points E_N of the projections F_N of F converge to $\bar{E}$. That is,* $\lim_{N\to\infty} E_N = \bar{E}$.

The closure of E is not necessary for our situation, since by the Bauer minimum principle we know the optimal solution to LP (6.3) is in E (*i.e.* at an extreme point of F).

As mentioned above, we will embed our projections in the infinite space somewhat differently than was done in (Cross et al., 1998). Let V^N be the optimal solution to the LP defined on the projection F_N. Note that in such an LP the

optimization is only over variables whose index is less than or equal to N. We will embed these solutions onto $\mathbb{R}^\infty$ by the following equation:

$$W_i^N = \begin{cases} V_i^N & \text{if } i \leq N \\ V_{max} & \text{otherwise} \end{cases}$$

We will now present a final lemma and then the main theorem of this chapter on which the algorithm will be based.

Lemma 6.3.10. *W^N is an extreme point of F.*

Proof. Assume not, by contradiction. Then there exist $x, y \in F$ such that $W^N = \frac{1}{2}(x+y)$. Consider this convex combination at each index: $W_i^N = \frac{1}{2}(x_i + y_i)$. Since $p_N(W^N) \in E_N$, $W_i^N = x_i = y_i$ for all $i \leq N$. Therefore, $W_i^N = V_{max} = \frac{1}{2}(x_i + y_i)$ for all $i > N$. Clearly we have either $x_i \geq V_{max} \geq y_i$ or $x_i \leq V_{max} \leq y_i$; without loss of generality assume the former. However, from inequality (2.2) we know that for all $f \in F$, $0 \leq f_i \leq V_{max}$ for all i. This is only possible if $x_i = V_{max}$, which implies $y_i = V_{max}$ for all i. Thus, $W^N = x = y$, proving the desired result. □

Theorem 6.3.11. *The sequence of embedded solutions W^N form a decreasing monotone sequence converging to V^*.*

Proof. By Lemma 6.3.10, we see that $W^N \in E_{N+1}$. Since W^{N+1} is the optimal extreme point in F_{N+1}, $W_i^{N+1} \leq W_i^N$ for all $1 \leq i \leq N$. Also, $W_i^n = W_i^{N+1} = V_{max}$ for all $i > N+1$. At index $N+1$ we have $W_{N+1}^N = V_{max} \geq V_{N+1}^{N+1} = W_{N+1}^{N+1}$. Since for all $f \in F$ we have $f \geq \theta$, we see that the sequence of W^Ns is bounded below. Thus, by the Monotone convergence principle, the W^Ns converge. The fact that

their limit is V^* follows from the fact that V^* is an extreme point minimizing LP (6.3). □

Note that a Hyper MDP constructed over an MDP with a finite number of actions and respecting assumption 2.1.1 is part of the general class of MDPs for which the above results apply. Thus, theorem 6.3.9 applies to the Hyper MDP case as well.

6.4 ABE: Anytime Bayesian Exploration

Consider an exploratory agent that requests an action choice from a black box at certain points in time. The time intervals between requests may vary, but the agent does not want to wait for long when it puts in a request. A simple example of this kind of situation can be posed in the context of the SysAdmin problem (see section 5.3.1): rather than dividing up the decision epochs into predetermined time slots, one can think of a decision epoch occurring every time the system administrator has to take a new action. If the system administrator is working on a machine, he cannot perform another action on another machine simultaneously, so the action choice can wait until he is done with the current machine.

The previous methods presented in this thesis output an action choice as soon as their computation is finished (which is dependent on the depth, number of samples selected and complexity of the problem). If the algorithm takes too long to choose an action, the agent will be left waiting idly; on the other hand, if the agent is not requesting an action, the algorithm will be left sitting idly until a new request is made. It would be beneficial to develop an algorithm that can return

an action choice essentially at any point in time. The more time between action requests, the better its response will be.

We present an algorithm based on the ideas presented in (Ghate et al., 2007). In contrast to our presentation, (Ghate et al., 2007) deals with the dual LP formulation. They fail to prove the absence of a duality gap which restricts one to solving only the dual formulation. Furthermore, the convergence of the value functions in (Ghate et al., 2007) is not guaranteed to be monotonic, which is needed in order to provide an anytime algorithm.

Our algorithm proceeds as follows. As in section 5.1 we construct a sparse tree with a specified number of samples per action (C) and a desired horizon (H). Rather than specifying large values of C and H to guarantee near-optimality, we start with reasonably small values of C and H in order to produce a relatively small sparse tree. The sparse tree is formulated as an LP (as in Chapter 4) and CPLEX is used to obtain a solution to this LP. Note that this solution is the value function for the sampled hyper states. This initial LP is our first projection, and thus we may refer to it as LP_1. If there is more time after solving LP_1, the next projection (LP_2) is constructed by either increasing the number of samples (δC) or the depth (δH) in the sparse tree of LP_1. LP_2 is thus an extension of LP_1 and thus, all of the results presented in section 6.3 apply, demonstrating that each successive projection will produce a better value function estimate. The algorithm continues in this fashion until a request for an action is received. At this point, the latest value function computed is used to select the optimal action. A detailed listing of the pseudocode for ABE can be found in Appendix A.

6.5 Experimental results

In this section we present empirical results for ABE when running experiments on the same domains as in section 4.4 and some of the problems from section 5.3.1. These results illustrate the improved performance when more time is allowed as well as demonstrate it compares favorably against other algorithms. Since ABE is essentially a generalization of *bayesExplore* (without the *rest* parameter), the plots for *bayesExplore* are not presented in the following results for clarity. In all of the results presented, ABE was started with 2 samples, a depth of 3 and incremented one sample at a time (*i.e.* $C = 2, H = 3, \delta C = 1, \delta H = 0$). The numbers in parentheses next to ABE in the legend indicate the amount of time in seconds given to ABE for action selection.

In figure 6–1 we ran ABE on the small MDP problem of section 4.4 and compared it against the performance of the LP approach with the parameter settings that yielded the best results. This is a very easy problem so there is little improvement when more time is given to ABE, as it achieves near-optimal results quickly. Nevertheless, it is able to achieve an improved performance compared to the best results that the LP approach was able to produce.

Figure 6–2 compares ABE against the best of the LP approach settings and R-MAX on the bandit problem of section 4.4. This problem highlights the advantage of allowing more time for action selection as well as the advantage of ABE over the other algorithms.

In the grid world problem of section 4.4 the performance of ABE was quite poor. This is due to the fact that the goal state cannot be "seen" by ABE unless

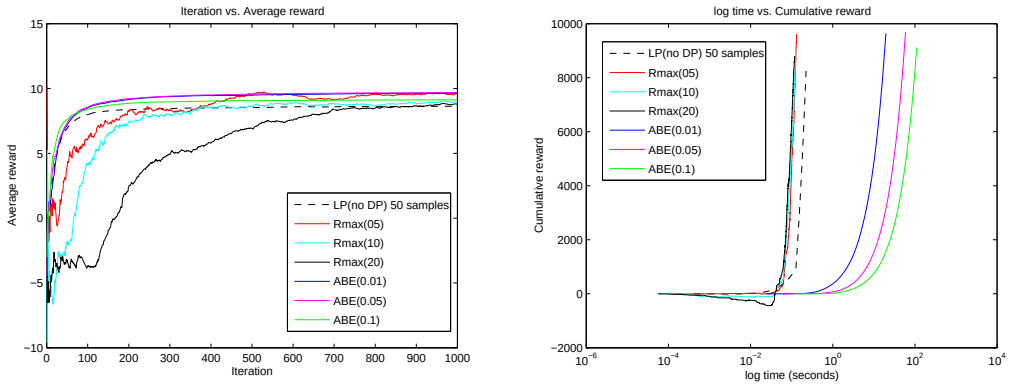

Figure 6–1: Small MDP: (a) Average reward versus iteration; (b) Cumulative reward versus log time

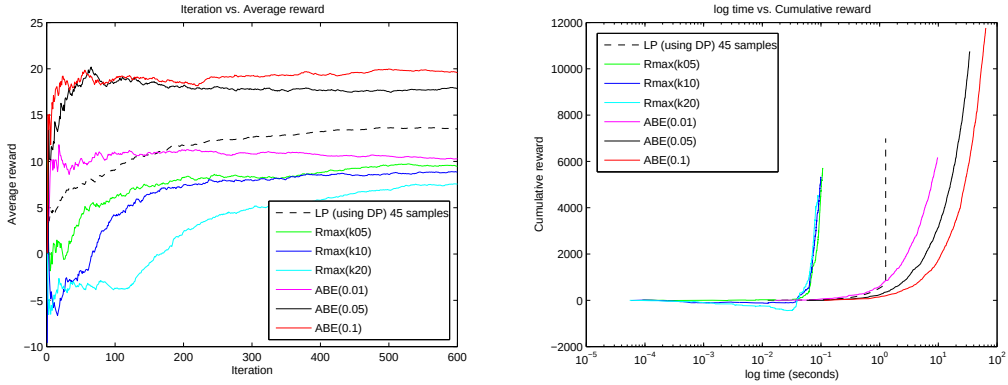

Figure 6–2: Bandit problem: (a) Average reward versus iteration; (b) Cumulative reward versus log time

we are able to grow the sparse tree to a sufficient depth. The advantage of the LP approach was that it maintained and used state-action values for exploration. This served as a form of "memory" for the algorithm by being able to estimate how close the states are to the goal. However, this method rendered it difficult to obtain theoretical results for the LP approach. Figure 6–3 demonstrates the performance of ABE on the grid world problem. Although the performance is

poor, we can still observe a slight improvement in performance when more time is allowed. The graphs from the LP approach are omitted since it is clear that it outperforms ABE in this problem.

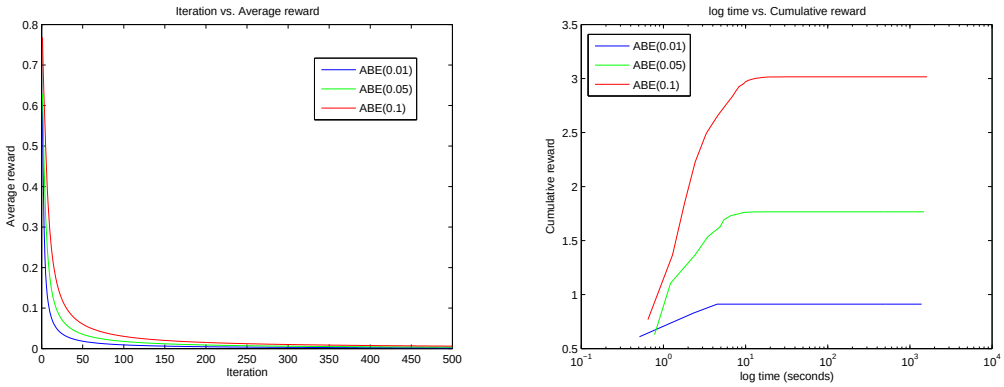

Figure 6–3: Grid world problem: (a) Average reward versus iteration; (b) Cumulative reward versus log time

The performance of the various algorithms on the Ring4 problem of section 5.3 is depicted in figure 6–4. As in the small MDP problem above, ABE achieves near-optimal performance very early on, so there is almost no improvement as more time is allowed. Nevertheless, its performance is still superior to the LP approach and R-MAX.

Figure 6–5 compares the various algorithms on the Legs_2^2 problem of section 5.3. We observe a similar behavior as before, further reassuring the validity of the method.

Finally, table 6–1 shows the savings obtained when using the solution of a previous projection as a starting point for the computation of a larger projection versus starting from scratch. We present the percentage of reduction in the

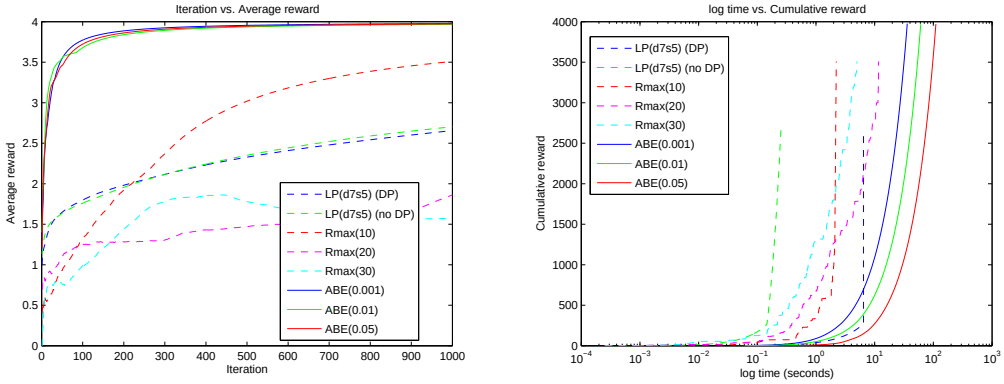

Figure 6–4: Ring4 problem: (a) Average reward versus iteration; (b) Cumulative reward versus log time

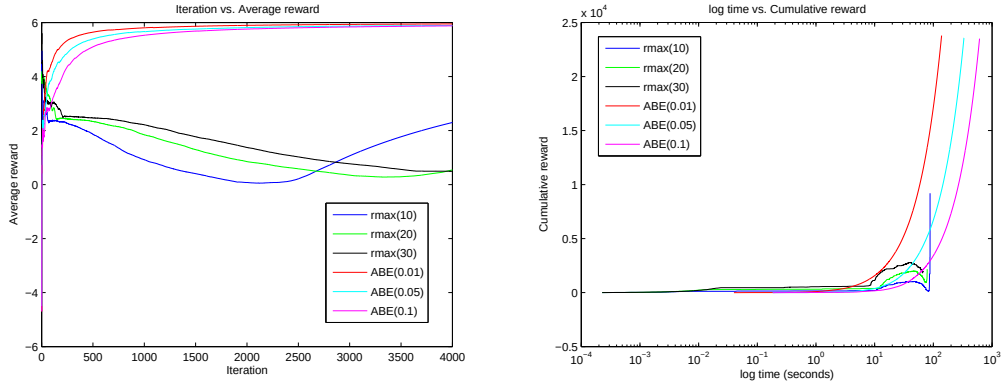

Figure 6–5: Legs_2^2: (a) Average reward versus iteration; (b) Cumulative reward versus log time

number of iterations and computation time taken by the CPLEX solver on each of the problems from section 4.4 as well as the Ring4 problem of section 5.3, averaged over 1000 iterations. We always started with $C = 2$ and $H = 3$, and tried different values of δC. The top column specifies the values of H and C being compared. It is evident that there are great savings when starting from an extreme point solution as opposed to computing the solution from scratch. In the first three

problems it is also clear that as we increase the number of samples, the savings increase. This is due to the low stochasticity of the problems.

Table 6–1: Percentage reduction on number of iterations/computation time

Problem	$H = 3, C = 3$	$H = 3, C = 4$	$H = 3, C = 5$	$H = 3, C = 6$
Small MDP	43.78%/33.33%	68.16%/50.49%	97.62%/81.12%	98.44%/83.91%
Bandit	49.89%/29.25%	64.39%/39.42%	84.30%/60.53%	87.97%/65.10%
Grid World	32.35%/19.57%	48.49%/26.36%	65.73%/42.47%	73.71%/49.14%
Ring4	26.82%/18.36%	47.27%/43.33%	35.27%/29.48%	36.55%/30.16%

CHAPTER 7
Conclusion

In this thesis we have presented three methods for computing exploration policies yielding high return. We have demonstrated the good empirical performance of our methods when compared against other state-of-the-art algorithms. In Chapter 4 we showed that by modifying the method of (Wang et al., 2005) we can achieve better results. The shortcoming of these two methods, however, is that it is difficult to obtain theoretical guarantees of their performance.

As a result, we presented the *bayesExplore* method in chapter 5 and proved that it can achieve near-optimal results independent of the size of the state space and problem representation. This result was heavily based on the results of (Kearns et al., 1999). As in the original paper, the required number of samples and horizon derived from the theory are still too restrictive to be applicable in practice. However, our empirical results demonstrated that even when limited to a small number of samples and a shallow horizon, *bayesExplore* achieves a superior performance to other algorithms.

The *ABE* algorithm presented in chapter 6 is the most significant contribution of this thesis. With this method we indirectly address the problem of the infeasibility of the requirements for near-optimality of *bayesExplore*. Rather than committing to a number of samples and horizon depth beforehand, we can increase these values gradually, proportional to the amount of time available, yet without

losing any previous computations. This method also simplifies the choice of the number of samples and horizon depth, as they will be dynamically set at each iteration. We proved formally and demonstrated empirically that the more time the algorithm is given to process, the better the action choice will be. Our theoretical results are not only applicable to Hyper MDPs, but to any general MDP with a countable number of states and bounded rewards. Furthermore, the duality result proved in theorem 6.2.3 is significant in its own right as it is the first time such a result has been shown (as far as the author's knowledge).

7.1 Future work

In chapter 5 we observed that the *rest* parameter can help speed up the algorithm without affecting the performance very much. It would be useful to obtain theoretical results regarding how the *rest* parameter affects the near-optimality results.

It is also clear that after a large number of iterations, the model estimate will be greatly improved, and thus, expanding all actions may not be necessary. At a certain point action-selection methods (Puterman, 1982; Even-Dar et al., 2006) could be used to reduce the size of the sampled tree. Results similar to what was presented in (Kearns and Singh, 1998) could be used to define the number of samples before action-selection can be used. Unlike the method in (Kearns and Singh, 1998), however, our methods might not try all actions from every state, and thus, we would not guarantee near-optimality of the model estimate.

One troublesome aspect is the "near-sightedness" of ABE. As was demonstrated by the grid world problem in section 6.5, if the goal is quite far away,

the sparse tree constructed by ABE may not be able to "see" the goal state in a reasonable amount of time. The LP approach got around this by maintaining and using state-action value pairs but we were not able to provide theoretical guarantees. A possible modification to ABE is to use state-action value pairs at the leaves of the sparse tree. However, this modification complicates the monotonicity guarantee so the theoretical results would have to be refined.

Finally, it would be interesting to extend the results of chapter 6 to MDPs with continuous state and action spaces. These types of MDPs are very common in control theory and robotics and would benefit greatly from an anytime method for online planning. Work on LPs in measure spaces (Kellerer, 1988; Lai and Wu, 1994) suggest that strong duality results can be obtained for general MDPs with continuous state and action spaces. However, the indexing of the increasing projections is no longer clear and the inductive arguments for the anytime algorithm do not immediately follow.

Appendix A

In this appendix we present the pseudocode for ABE. Before doing so, we will define the projection procedure (see algorithms 5 and 6) which, given a desired depth H, a specified number of samples C, and a sampled finite hyper tree to start from (T), constructs a projection of the infinite dimensional hyper tree. We assume that the hyper tree T is a data structure where $T.root$ returns the hyper state at the root of the tree (*i.e.* a pair of the form (s, ψ)), $T.numSamples$ returns the number of samples for each action from the root, $T.isLeaf$ returns whether T is a leaf node, $T.subtree(a, i)$ points to the subtree obtained at the ith sample of action a from the root, and $T.nextStates$ is the set of sampled next states from the root. This procedure is used by our main algorithm to create the initial finite LP, and then to obtain the increasing projections. Algorithm 7 introduces ABE (Anytime Bayesian Exploration). It receives as parameter an initial depth H, initial number of samples C, depth increment δH and samples increment δC. The initial depth and samples should be chosen so as to guarantee that the computation time with them will be close to the smallest expected time interval between action requests. Once the initial solution has been found, the algorithm will continue to increment the size of the projection, alternating between increasing the number of samples and increasing the depth. As soon as a request is received, the algorithm interrupts its current processing and returns the latest solution available. Theorem 6.3.11 guarantees that each successive computation produces an improved solution.

Algorithm 5 Projection(T, H, C)

```
1: if H == 0 then
2:   return T
3: end if
4: if T.isLeaf then
5:   for a = 1 to |A| do
6:     for i = 1 to C do
7:       T ← addSample(T, H − 1, C)
8:     end for
9:   end for
10: else
11:   if T.numSamples == C then
12:     for a = 1 to |A| do
13:       for i = 1 to C do
14:         T.subtree(a, i) ← Projection(T.subtree(a, i), H − 1, C)
15:       end for
16:     end for
17:   else
18:     for a = 1 to |A| do
19:       for i = 1 to T.numSamples do
20:         T.subtree(a, i) ← Projection(T.subtree(a, i), H − 1, C)
21:       end for
22:       for i = T.numSamples + 1 to C do
23:         T ← addSample(T, H − 1, C)
24:       end for
25:     end for
26:   end if
27: end if
28: return T
```

Algorithm 6 addSample(T, H, C)

if $H == 0$ **then**
  return T
end if
$(s, \psi) \leftarrow T.root$
for $a = 1$ to $|\mathcal{A}|$ **do**
  Generate a sample from $\mathbf{p}_s^a(\psi)$. Let s' be this next state.
  $T.numSamples \leftarrow T.numSamples + 1$
  $T.isLeaf \leftarrow FALSE$
  $T.nextStates \leftarrow T.nextStates \cup s'$
  $T.subtree(a, T.numSamples).root \leftarrow (s', D_{s,s'}^a(\psi))$
  $T.subtree(a, T.numSamples).numSamples \leftarrow 0$
  $T.subtree(a, T.numSamples).isLeaf \leftarrow TRUE$
  $T.subtree(a, T.numSamples).nextStates \leftarrow \emptyset$
  $T.subtree(a, T.numSamples) \leftarrow Projection(T.subtree(a, i), H - 1, C)$
end for
return T

Algorithm 7 ABE($MDP, s_0, \psi, H, C, \delta H, \delta C$)

$s \leftarrow s_0$
$T \leftarrow \emptyset$
$T.root \leftarrow (s_0, \psi)$
$T.nextStates \leftarrow \emptyset$
$T.numSamples \leftarrow 0$
while Agent is exploring **do**
  $T \leftarrow Projection(T, H, C)$
  Construct an LP from T in the usual manner.
  Solve the constructed LP. Let $\bar{V}$ be the solution vector found.
  $a \leftarrow \arg\max_a \left[\bar{r}^a_s(\psi) + \gamma \sum_{s' \in T.nextStates} \bar{p}^a_{s,s'}(\psi) \bar{V}(s', D^a_{s,s'}(\psi))\right]$
  while An action request interrupt has not been received **do**
    $H \leftarrow H + \delta H$
    $C \leftarrow C + \delta C$
    $T \leftarrow Projection(T, H, C)$
    Construct an LP from T in the usual manner.
    Solve the constructed LP, using $\bar{V}$ as the starting extreme point. Let $\bar{V}$ be the new solution vector found.
    $a \leftarrow \arg\max_a \left[\bar{r}^a_s(\psi) + \gamma \sum_{s' \in T.nextStates} \bar{p}^a_{s,s'}(\psi) \bar{V}(s', D^a_{s,s'}(\psi))\right]$
  end while
  Return action a to agent, and observe transition $(s \mapsto s')$ under action a performed by agent
  if $s' \in T.nextStates$ **then**
    Let i be the sample number for which hyper state $(s', D^a_{s,s'}(\psi)$ is the root.
    $T \leftarrow T.subtree(a, i)$
    Discard all the elements of $\bar{V}$ not in $T.subtree(a, i)$, as they are no longer needed.
  else
    $T \leftarrow \emptyset$
    $T.root \leftarrow (s', D^a_{s,s'}(\psi))$
    $T.nextStates \leftarrow \emptyset$
    $T.numSamples \leftarrow 0$
  end if
  $s \leftarrow s'$
end while

Bibliography

Albus, J.: 1981, *Brains, Behaviour, and Robotics*, BYTE publications, Peterborough, N.H.

Aliprantis, C. D. and Border, K. C.: 2006, *Infinite Dimensional Analysis: A Hitchiker's Guide, Third Edition*, Springer-Verlag, New York

Anderson, E. and Nash, P.: 1987, *Linear Programming in Infinite-Dimensional Spaces: Theory and Applications*, John Wiley & Sons Ltd., New York

Bellman, R.: 1957, *Dynamic Programming*, Princeton University Press, Princeton, New Jersey

Bellman, R. and Kalaba, R.: 1959, On adaptive control processes, in *IRE Trans.*, Vol. 4, pp 1–9

Bertsekas, D. and Tsitsiklis, J.: 1996, *Neuro-Dynamic Programming*, Athena Scientific

Boutilier, C., Dearden, R., and Goldszmidt, M.: 1995, Exploiting structure in policy construction, in C. Mellish (ed.), *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp 1104–1111, Morgan Kaufmann, San Francisco

Brafman, R. I. and Tennenholtz, M.: 2001, R-MAX — a general polynomial time algorithm for near-optimal reinforcement learning, in *IJCAI*, pp 953–958

Bryant, R. E.: 1986, Graph-based algorithms for boolean function manipulation, *IEEE Transactions on Computers* **35(8)**, 677

Castro, P. S. and Precup, D.: 2007, Using linear programming for bayesian exploration in markov decision processes, in *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp 2437–2442

Cherkassky, V. and Mulier, F.: 1996, Comparison of adaptive methods for function estimation from samples, in *IEEE Transactions on Neural Networks*, pp 969–984

Chvátal, V.: 1983, *Linear Programming*, W.H. Freeman and Company, New York

Cross, W. P., Romeijn, H. E., and Smith, R. L.: 1998, Approximating extreme points of infinite dimensional convex sets, *Mathematics of Operations Research* **23(2)**, 433

de Farias, D. and Roy, B. V.: 2001, On constraint sampling for the linear programming approach to approximate dynamic programming, in *Mathematics of Operations Research*

de Farias, D. and Roy, B. V.: 2003, The linear programming approach to approximate dynamic programming, in *Operations Research*

Dean, T. and Kanazawa, K.: 1989, A model for reasoning about persistence and causation, *Artificial Intelligence* **5(3)**, 142

Dearden, R., Friedman, N., and Andre, D.: 1999, Model based bayesian exploration, in *Proceedings of Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp 150–159

Duda, R. O., Hart, P. E., and Stork, D. G.: 2000, *Pattern Recognition, Second edition*, John Wiley & Sons Ltd., New York

Duff, M. O.: 2002, *Ph.D. thesis*, University of Massachussets

Even-Dar, E., Mannor, S., and Mansour, Y.: 2006, Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems, *Journal of Machine Learning Research* **7**, 1079

Ghate, A., Sharma, D., and Smith, R. L.: 2007, A shadow simplex method for infinite linear programs, *Submitted to Mathematical Programming*

Gittins, J. and Jones, D.: 1979, Bandit processes and dynamic allocation indices, *Journal of the Royal Statistical Society, Series B* **41(2)**, 148

Guestrin, C., Koller, D., and Parr, R.: 2001, Max-norm projections for factored MDPs, in *IJCAI*, pp 673–682

Guestrin, C., Patrascu, R., and Schuurmans, D.: 2002, Algorithm-directed exploration for model-based reinforcement learning in factored mdps, in *ICML*, pp 235–242

Hauskrecht, M. and Kveton, B.: 2004, Linear program approximations for factored continuous-state markov decision processes, in *Advances in Neural Information Processing Systems*, MIT Press, Cambridge, MA

Hernández-Lerma, O. and Lasserre, J. B.: 1996, Average optimality in markov control processes via discounted-cost problems and linear programming, *SIAM Journal on Control and Optimization* **34(1)**, 295

Hernández-Lerna, O.: 1989, *Adaptive Markov Control Processes*, Springer-Verlag, New York

Hernández-Lerna, O. and Lasserre, J. B.: 2002, The linear programming approach, in E. A. Feinberg and A. Shwartz (eds.), *Handbook of Markov Decision Processes*

Hoey, J., St-Aubin, R., Hu, A., and Boutilier, C.: 1999, SPUDD: Stochastic planning using decision diagrams, in *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pp 279–288

Hordijk, A.: 1994, Linear programming formulation of mdps in countable state space: The multichain case, *Mathematical Methods of Operations Research* **40(1)**, 91

Howard, R.: 1966, Information value theory, *IEEE Transactions on Systems Science and Cybernetics* **2**, 22

Kaelbling, L.: 1993, *Learning in Embedded Systems*, MIT Press

Kearns, M. and Singh, S.: 1998, Near-optimal reinforcement learning in polynomial time, in *Proc. 15th International Conf. on Machine Learning*, pp 260–268, Morgan Kaufmann, San Francisco, CA

Kearns, M. J. and Koller, D.: 1999, Efficient reinforcement learning in factored MDPs, in *IJCAI*, pp 740–747

Kearns, M. J., Mansour, Y., and Ng, A. Y.: 1999, A sparse sampling algorithm for near-optimal planning in large markov decision processes, in *IJCAI*, pp 1324–1231

Kellerer, H.: 1988, Measure theoretic versions of linear programming, *Mathematische Zeitschrift* **198**, 367

Klabjan, D. and Adelman, D.: 2006, A convergent infinite dimensional linear programming algorithm for deterministic semi-markov decision processes on borel spaces, in *To appear in Mathematics of Operations Research*

Koller, D. and Parr, R.: 2000, Policy iteration for factored MDPs, in *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pp 326–334

Lai, H. and Wu, S.: 1994, Linear programming in measure spaces, *Optimization* **29**, 141

Lasserre, J. B.: 1992, Average optimal stationary policies and linear programming in countable space markov decision processes, in *Proceedings of the 31st Conference on Decision and Control*, pp 3325–3327

Martin, J.: 1967, *Bayesian Decision Problems and Markov Chains*, John Wiley & Sons, New York

Munkres, J.: 2000, *Topology (2nd edition)*, Prentice Hall

Puterman, M. L.: 1982, Action elimination procedures for modified policy iteration algorithms, *Operations Research* **30**, 301

Puterman, M. L.: 1994, *Markov Decision Processes*, John Wily & Sons, New York

Ratitch, B.: 2004, *Ph.D. thesis*, McGill University

R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi: 1993, Algebraic Decision Diagrams and Their Applications, in *IEEE /ACM International Conference on CAD*, pp 188–191, IEEE Computer Society Press, Santa Clara, California

Russell, S. and Norvig, P.: 1995, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Inc., Upper Saddle River, New Jersey

Schweitzer, P. and Seidmann, A.: 1985, Generalized polynomial approximations in markovian decision processes, *J. Math. Anal. Appl.* **110**, 568

Somenzi, F.: 1998, Cudd: Cu decision diagram package release

Strehl, A. and Littman, M.: 2004, An empirical evaluation of interval estimation for markov decision processes, in *The 16th IEEE International Conference on Tools with Artificial Intelligence*, pp 128–135

Strehl, A. and Littman, M.: 2005, A theoretical analysis of model-based interval estimation, in *International Conference on Machine Learning*

Sutton, R. S. and Barto, A. G.: 1998, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Massachusetts

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

Wang, T., Lizotte, D., Bowling, M., and Schuurmans, D.: 2005, Bayesian sparse sampling for on-line reward optimization, in *International Conference on Machine Learning*

Wiering, M. and Schmidhuber, J.: 1998, Efficient model-based exploration, in *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, pp 223–228

Key to Abbreviations

ABE: Anytime Bayesian Exploration

ADD: Algebraic Decision Diagram

AI: Artificial Intelligence

ALP: Approximate Linear Programming

BDD: Binary Decision Diagram

CPT: Conditional Probability Table

DAG: Directed, acyclic graph

DBN: Dynamic Bayesian Network

DP: Dynamic Programming

IE: Interval Estimation

LP: Linear Programming

MDP: Markov Decision Process

RL: Reinforcement Learning

tvs: Topological Vector Space