Using Linear Programming for Bayesian Exploration in Markov Decision Processes

Pablo Samuel Castro and Doina Precup
McGill University
School of Computer Science
{pcastr,dprecup}@cs.mcgill.ca

Abstract

A key problem in reinforcement learning is finding a good balance between the need to explore the environment and the need to gain rewards by exploiting existing knowledge. Much research has been devoted to this topic, and many of the proposed methods are aimed simply at ensuring that enough samples are gathered to estimate well the value function. In contrast, [Bellman and Kalaba, 1959] proposed constructing a representation in which the states of the original system are paired with knowledge about the current model. Hence, knowledge about the possible Markov models of the environment is represented and maintained explicitly. Unfortunately, this approach is intractable except for bandit problems (where it gives rise to Gittins indices, an optimal exploration method). In this paper, we explore ideas for making this method computationally tractable. We maintain a model of the environment as a Markov Decision Process. We sample finite-length trajectories from the infinite tree using ideas based on sparse sampling. Finding the values of the nodes of this sparse subtree can then be expressed as an optimization problem, which we solve using Linear Programming. We illustrate this approach on a few domains and compare it with other exploration algorithms.

1 Introduction

A key problem in reinforcement learning is posed by the need to explore the environment sufficiently in order to discover the sources of reward, while at the same time exploiting the knowledge that the agent has already, by taking actions that yield high return. The most popular techniques in the current literature (e.g.,ϵ-greedy or Boltzmann exploration) are not aimed at efficient exploration; instead, they ensure that action choices are randomized, which enables the agent to try out all actions in all states. This approach guarantees the convergence of reinforcement learning algorithms, but is not efficient from the point of view of the number of samples gathered, and may cause the agent to repeatedly try harmful actions. An extensive line of research has been devoted to directed exploration methods, most of which are based on heuristics that help the agent get data while trying to protect it from "danger", e.g. [Thrun, 1992; Meuleau and Bourgine, 1999]. Recently, several algorithms have been proposed which carry guarantees in terms of the number of samples necessary for the agent to attain almost optimal performance [Kearns and Singh, 1998; Brafman and Tennenholtz, 2001; Strehl and Littman, 2005; Strehl *et al.*, 2006].

In this paper we explore a Bayesian approach to exploration. The initial idea was due to Bellman [1959] who suggested keeping information about the agent's current state of knowledge, in addition to the model being learned by the agent. This method is Bayes-optimal from the point of view of decision making, but its complexity grows exponentially with the horizon considered. This makes it inapplicable except for the case of bandit problems, where it gives rise to Gittins indices. Recently, a body of work on Bayesian reinforcement learning has developed method for approximating the Bayes-optimal procedure for the case of general MDPs [Dearden *et al.*, 1999; Duff, 2002; Wang *et al.*, 2005]. In this paper, we pursue a similar idea, but with two important differences. First, we propose to use linear programming in order to approximate the value function during this procedure. Second, we use a sampling approximation based on the value function to expand the horizon of the decision making process.

The paper is organized as follows. In Section 2 we present the necessary background. In Section 3 we present the linear programming approach to this problem. In Section 4 we present empirical results in three domains.

2 Background

A finite Markov Decision Process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the finite state space, $\mathcal{A}$ is the finite action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \Re$ defines the transition probabilities, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward function, bounded by R_{MAX}. If the agent is in state $s \in \mathcal{S}$ and performs action $a \in \mathcal{A}$, $\mathcal{T}(s, a, \cdot)$ is the distribution over next possible states and $\mathcal{R}(s, a)$ is the expected immediate reward. A deterministic stationary policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is a function that determines what action to take depending on the current state. One of the goals of reinforcement learning is to find the policy that maximizes the expected discounted return received, given by $\sum_{t=1}^{\infty} \gamma^{t-1} r_t$, where r_t is the reward received at time step t

and $\gamma \in (0,1)$ is a discount factor. The *value* of a state $V^\pi(s)$ is given by the expected discounted return received when executing policy π starting from state s:

$$V^\pi(s) = E_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \right] \quad (1)$$

Note that the upper bound on the value of any policy from any state is $R_{MAX}/(1-\gamma)$

2.1 Hyperstate MDPs

In this paper we consider MDPs in which the transition probabilities $\mathcal{T}$ and rewards $\mathcal{R}$ are unknown. In this situation, the agent maintains an estimate of the MDP model based on prior knowledge and its observations. For the transition probabilities, it is convenient to use a distribution that is closed under updates performed after observing a state transition. For MDPs with finite state and action sets, the estimate of the transition probabilities can be maintained in a matrix $\mathcal{P}$ of size $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$. As shown in [Martin, 1967], the Matrix Beta distribution is closed under updates and is a natural choice of distribution for this case. We denote by $\mathcal{M} = \left[m_{ij}^a\right]$ the current indexing parameters of the distribution, where m_{ij}^a is simply the number of observed transitions from state i to state j under action a. The expected transition probabilities based on this model are given by $E\left[p_{ij}^a\right] = \bar{p}_{ij}^a = m_{ij}^a / \sum_{k=1}^{|\mathcal{S}|} m_{ik}^a$. Letting the matrix $N = \left[n_{ij}^k\right]$ denote an additional number of observed transitions from state i to state j under action a, in [Martin, 1967] it is shown that the posterior distribution $\mathcal{P}'$ with parameters $\mathcal{M}' = \mathcal{M} + N$ is also a Matrix Beta distribution. In what follows, we will denote by $D_{ij}^a(\mathcal{M})$ the new distribution obtained from $\mathcal{M}$, through this operation, after observing on transition from state i to state j under action a.

The estimated reward received when performing action a from state i can then be estimated by $\bar{r}_i^a = \sum_{j=1}^{|\mathcal{S}|} \bar{p}_{ij}^a \tilde{r}_{ij}^a$, where $\tilde{r}_{ij}^a$ is the average reward obtained when going for state i to state j under action a. These rewards can be summarized in a matrix $\bar{\mathcal{R}}$.

The Matrix Beta distribution can be viewed as a summary of the information that the agent has accumulated so far. This information, together with the original MDP states and actions, will form a new, *Hyperstate-MDP* (HMDP). The actions in the HMDP are the same as in the original MDP. The states of the HMDP consist of pairings of states from $\mathcal{S}$ with possible information states $\mathcal{M}$. A hyperstate $(i, \mathcal{M}, \bar{\mathcal{R}})$ contains an MDP state i, all the counts summarizing the agent's experience so far $\mathcal{M}$, and all the estimates of the expected rewards $\bar{\mathcal{R}}$. The counts define a distribution over all MDP models consistent with the data observed so far.

At any given time step when action a is taken in the original MDP from state i, precisely one transition is observed, to some state j. So, from any given hyperstate $(i, \mathcal{M}, \bar{\mathcal{R}})$, we can consider taking all possible actions $a \in \mathcal{A}$. If such an action results in a state j, the new hyperstate will be $(j, D_{ij}^a(\mathcal{M}), D_{ij}^a(\bar{\mathcal{R}}))$, where we have updated the innformation model as described above to reflect the new transition. We note that this update only affects the parameters for state i; all parameters for other states do not change. Assuming that there are a fixed number of rewards R that can be observed, each hyperstate in the HMDP has at most $|\mathcal{A}| \times |\mathcal{S}| \times R$ successors. Moreover, each successor hyperstate is uniquely indexed. Hence, the HMDP is an infinite tree. Note also that the HMDP takes into account all possible transitions. Hence, it is fully known, even though the MDP itself is not.

One can express the Bellman optimality equations for an HMDP as:

$$V(i, \mathcal{M}, \bar{\mathcal{R}}) = \max_{a \in \mathcal{A}} \left\{ \bar{\mathcal{R}}_i^a + \gamma \sum_j \bar{p}_{ij}^a(\mathcal{M}) V(j, D_{ij}^a(\mathcal{M}), D_{ij}^a(\bar{\mathcal{R}})) \right\}$$

Solving this set of equations exactly would yield an optimal policy [Bellman and Kalaba, 1959; Martin, 1967]. However, it is clear that the number of states in the HMDP is infinite, so an exact solution is intractable. It is also clear that the number of hyperstates increases exponentially with the depth of the tree, so an exact solution of a finite-horizon subset of the tree would be limited to very shallow depths. The focus of this paper is to present an algorithm for computing an empirically *good* learning policy from a finite subset of the tree. Theoretical guarantees will be left for future work and we will focus on comparing the performance of our algorithm to other well-known exploration techniques.

2.2 Related Work

Optimal learning has seen an increase in interest in recent years. Much of the recent work has been spurred by [Kearns and Singh, 1998], where the authors introduce E^3, an algorithm that is guaranteed to converge to an optimal learning policy in polynomial time. The drawbacks with this algorithm are its difficulty of implementation, and the intuition that it does not necessarily scale well to large state spaces. A practical model-based algorithm with similar guarantees was given in [Brafman and Tennenholtz, 2001]. More advanced model-based algorithms with PAC guarantees, based on real-time dynamic programming, have been recently proposed in [Strehl and Littman, 2005; Strehl *et al.*, 2006]. These methods explore by assigning a reward for exploratory actions. The drawback of both [Kearns and Singh, 1998] and [Strehl *et al.*, 2006] is that their strategies are myopic - they do not consider the long term effects their actions may have on the total reward.

Using hyperstates as the model for exploration overcomes this myopic behavior. The concept of hyperstates was first introduced in [Bellman and Kalaba, 1959], which refers to this model as an *adaptive control process*. Although mathematically rich, the paper presents no algorithmic approach for this idea. In [Duff, 2002], various heuristic methods are presented for approximating an exact solution to the adaptive control process. These produce empirically good results, but with no general theoretical guarantees.

Wang et al [2005] propose sampling from the infinite hypertree to produce a small, more manageable tree. They solve exactly the MDPs along the path they are sampling, which gives them estimates for the values of different actions. Then, they use Thompson sampling to expand the tree locally under promising actions. The authors estimate unsampled regions by effectively 'copying' the last sampled node down to the

current planning horizon. Finally, sparse sampling [Kearns *et al.*, 1999] is used to decide what action is optimal. A correction procedure is used when the desired horizon has not been reached. Empirical results demonstrate that the algorithm performs well in comparison to other algorithms. Furthermore, the sparse sampling technique enables the algorithm to be able to handle infinite state and action spaces.

3 Solving the hyperstate MDP

In this paper we take an approach similar to [Wang *et al.*, 2005] but with two key differences. First, action-value estimates are maintained for the state-action pairs of the original MDP, in addition to hyperstate values. The action-values are updated from the obtained samples using standard Q-learning [Sutton and Barto, 1998]. Hence, we always have an easy estimate of how good different actions are. Second, we compute the value of the hyperstates using Linear Programming. Linear Programming (LP) is a technique that has been around for a long time, is well understood and has an elegant theory [Puterman, 1994]. Furthermore, recent work in approximate linear programming [de Farias and Roy, 2003; 2004; Hauskrecht and Kveton, 2004]) suggests that this approach can be used to work with continuous states using linear function approximation. We are hopeful that such techniques could eventually be used to generalize our approach to continuous MDPs. However, for now we limit ourselves to discrete MDPs.

In a manner similar to [Schweitzer and Seidmann, 1985], the optimality equations for the hyperstate MDP can be formulated as a linear program as follows:

$$\text{minimize} \qquad \sum_i V(i, \mathcal{M}, \bar{\mathcal{R}})$$

such that

$$V(i, \mathcal{M}, \bar{\mathcal{R}}) - \left[\bar{\mathcal{R}}_i^a + \gamma \sum_j \bar{p}_{ij}^a(\mathcal{M}) V(j, D_{ij}^a(\mathcal{M}), D_{ij}^a(\bar{\mathcal{R}})) \right] \geq 0$$

for all states $i \in \mathcal{S}$, all actions $a \in \mathcal{A}$ and all information states $\mathcal{M}$. We will refer to this formulation as the *exact LP*. However, at a depth d, the number of hyperstates is at least $\mathcal{O}\left((|\mathcal{S}| \times |\mathcal{A}|)^d\right)$ (more if we consider several possible reward levels). This means that for depth d the linear program has at least $\mathcal{O}\left((|\mathcal{S}| \times |\mathcal{A}|)^d\right)$ variables and constraints, a prohibitive amount.

In [Kearns *et al.*, 1999] the authors overcome a similar obstacle by constructing a sparse sample tree. We apply a similar idea here and construct a sampling tree incrementally. More precisely, we sample from the hyperstate MDP using our current estimates of the action values to decide what to do. At each node, we use the corresponding estimate of the dynamics of the system to sample the next node in the trajectory. Each trajectory is completed to a desired horizon. We continue constructing trajectories in this way until a maximum desired number of samples has been reached.

If we encounter a hyperstate for which a value was computed already using an LP, we use the LP-based estimate (which, in the limit, approaches the true optimal value of the corresponding MDP state). If such a value has not been computed, the action-value function for the MDP state is used instead. Note that any unsampled regions constitute a subtree of our hyperstate MDP. These unsampled subtrees are estimated by setting the value of the hyperstate at the root of the subtree, $(i, \mathcal{M}, \mathcal{R})$ to the value of the correspondinng MDP state i. With this method, we can effectively choose how many variables and constraints we want to use in the linear program.

Once the sampled hyper tree is built, we can solve it using linear programming. Then, in order to gather more samples, we will choose action greedily based on the values computed by the linear program. If we enter an unsampled region, the action choice becomes greedy with respect to the value estimates attached to the original MDP states, until we decide to construct a new hyper tree and compute new values.

Algorithm 1 Explore($levels, maxSamples, numSteps, maxEpochs$)

```
Initialize the matrix of counts to all 1s (uniform distribution)
while epochs ≤ maxEpochs do
    Construct a hyper-tree of a depth of levels using maxSamples sampled trajectories
    Solve the LP of the sample hypertree
    Choose action greedily based on hyperstate values
    Observe transition and update hyper parameter for observed transition
    for i = 1 to numSteps do
        Choose action greedily based either on current hyperstate value or current estimate of state values
        Observe transition and update hyper parameters based on the observed transition
    end for
    epochs ← epochs + numSteps1
end while
```

Algorithm 1 presents an outline for our approach. The $maxEpochs$ parameter defines how many total samples will be gathered from the environment. The $levels$ parameter describes the depth to which we will expand the sampled hypertree. The $maxSamples$ parameter controls how many trajectories are sampled (i.e., the width of the tree). Together, these two parameters control how large each linear program is. The $numSteps$ parameter defines how many steps in the environment are taken before the LP solution is recomputed. This allows us to trade off the precision of the computation against time. Obviously, if we re-compute the LP after every sample, we always generate samples based on the action that is truly believed to be best. If we wait longer, we may start selecting actions based on imperfect estimates of the values of the underlying MDP. However, this speeds up the computation significantly.

4 Empirical results

We tested our algorithm on three different problems. For all domains, results are averaged over 30 independent runs. We compared our approach with a Q-learning agent (with $\alpha = 0.1$) using an ϵ-greedy approach (with varying values of ϵ),

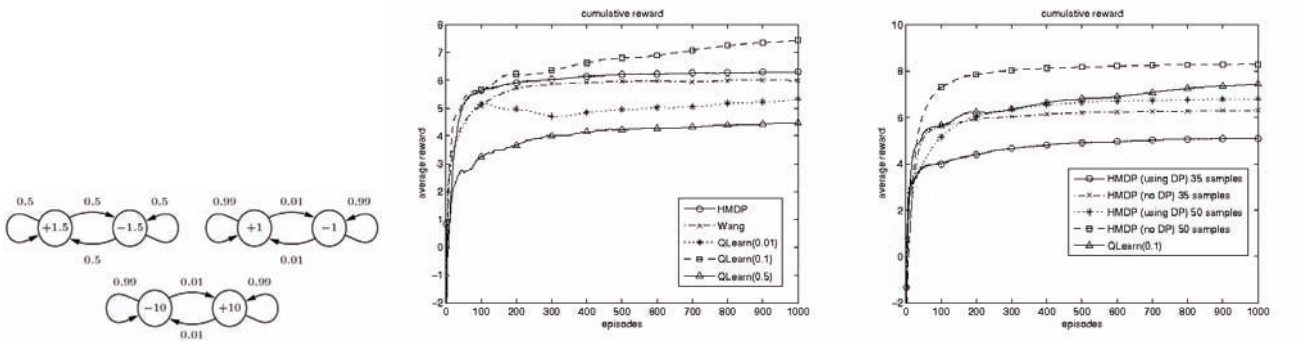

Figure 1: Two-state, three-action MDP: dynamics (left, comparison with the other algorithm (center) and effect of parameters (right)

and with the algorithm described in [Wang *et al.*, 2005]. We also experimented with a myopic agent (acting only based on immediate reward), and against an agent which chooses actions randomly. These last two algorithms did far worse than all the others, and so are omitted here for clarity. For our algorithm, we experimented with different parameter settings, as explained in more detail below. Because the primal LP has more constraints than variables, the dual was solved instead.

We also considered two different possible uses of the model that is constructed by our algorithm. In the first version, once the desired number of epochs has been completed, we perform dynamic programming using the acquired model, to determine a value function for the original MDP. In the second setting, We just use the values determined by the LP as value estimates. Obviously, this latter approach is faster, since no extra computation is required, but it is also more imprecise.

Small MDP

The first problem was a two-state, three-action MDP, described in the left panel of Figure 3. Intuitively, the best policy is to perform action 1 until state 2 is reached, then do action 3 and keep in the same state.

The center graph compares our algorithm, with parameters $levels = 7$,$numSteps = 6$,$maxSamples = 35$, with Q-learning using three different values of ϵ (0.01, 0.1 and 0.5) and the approach by Wang et al. For all algorithms we plot average return per episode. Note that the implementation of the algorithm described in [Wang *et al.*, 2005] was not able to handle more than 35 samples, which is why we performed the comparison at this level. Although Q-Learning with $\epsilon = 0.1$ outperformed all the other algorithms in this graph, we can see that if we allow our algorithm 50 samples to construct the hypertree, it performs better than the rest. Interestingly, in this case, using the values from the LP also gives better performance than doing one last step of dynamic programming.

Bandit problem

The second problem is similar to a bandit problem with two slot machines (bandits) and three actions. This problem was modeled using three states and three actions, with dynamics as given in figure 4. The dynamics are shown in the left panel for action 1 (solid), action 2 (dashed) and action 3 (solid grey). The leftmost state corresponds to not winning, the middle state corresponds to winning under machine 1 and the rightmost state corresponds to winning under machine 2. There is no cost associated with gambling.

For the comparison between algorithms, we use parameters $levels = 6$,$numSteps = 5$,$maxSamples = 45$. For this problem [Wang *et al.*, 2005]'s algorithm was able to handle up to 45 samples as well. Our algorithm outperformed all the others. In Figure 4 we can see that even with only 15 samples, our algorithm does almost as well as the best Q-learning algorithm.

Grid World

The third problem is a 2x4 gridWorld with a reward of $+1$ in the top right state and 0 everywhere else. There are four actions (north, south, west and east) with a 0.1 probability of remaining in the same state. If the agent tries to move into a wall it will deterministically stay in the same state. The parameter values used are $levels = 9$,$numSteps = 8$,$maxSamples = 15$.

This is a larger problem than the first two, and all the other algorithms are 'stuck' with low reward, while our algorithm is able to find a good policy. In figure **??** it is clear that using Dynamic Programming to obtain state value estimates is advantageous for this task.

It is interesting to see that in the first two problems using the values returned by the LP to estimate the state values yields better returns initially. This is probably because of the 'lookahead' nature of HMDP. However, for the gridWorld problem the performance is much better when using the DP solution to estimate the state values. This behavior is most probably due to the size of the hypertree created. As the state and action space get bigger, more samples are needed to obtain state value estimates that are closer to the true values. It should be noted that even when the DP was not solved exactly, in all cases the performance when using HMDPs was superior to the other algorithms. We note that we also experimented with Boltzmann exploration algorithms (omitted from the plots for clarity), but in all cases their performance was similar to the Q-learning algorithms.

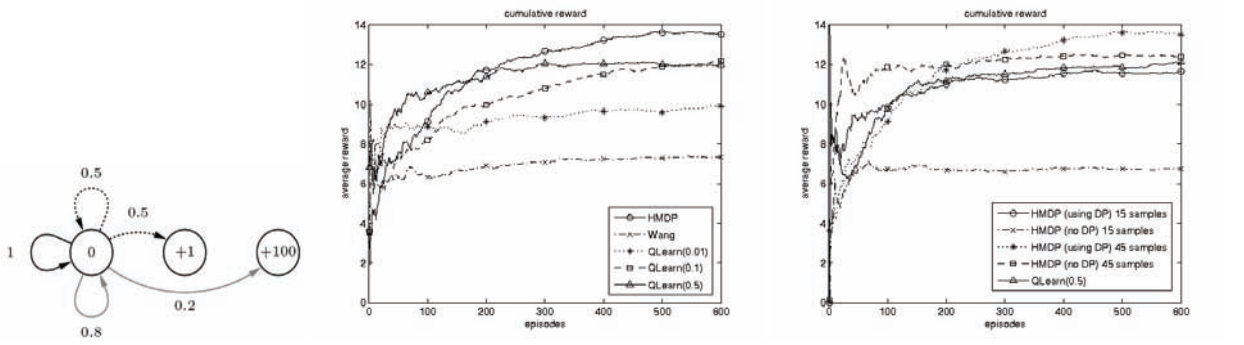

Figure 2: Bandit problem: dynamics (left), comparison of different algorithms (center) and parameter influence (right)

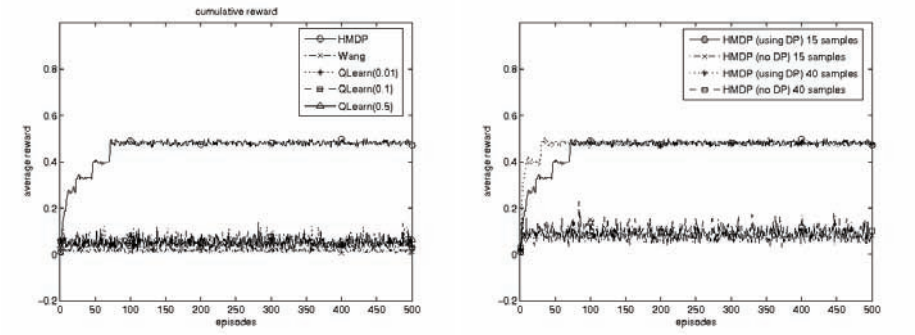

Figure 3: Comparison of algorithms (left) and influence of parameters (right) in the gridworld task

Running time

The results demonstrate that our approach has an advantage compared to the other methods in terms of accuracy. However, in terms of computation time, using hyperstates is obviously much slower than Q-learning. Table 1 plots the average running time per episode when solving the DP to estimate the state values versus using the values returned by the LP for the different domains, as well as the running time of [Wang *et al.*, 2005]'s algorithm. It is worth observing the difference in performance/running time depending on how the state values are estimated (solving with DP or using the values returned by the LP). Although our algorithm is considerably slower when solving with DP, it is faster than [Wang *et al.*, 2005]'s algorithm when using the values returned by the LP solution. However, for the gridWorld problem, although solving with HMDP is slower than [Wang *et al.*, 2005]'s algorithm, the savings when using [Wang *et al.*, 2005]'s algorithm are insignificant when compared against the superior performance demonstrated in figure 3.

	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 1: Comparison of running times

5 Conclusions and future work

The aim of this paper was to demonstrate the performance of Bayesian learning using sparse sampling and linear programming. It was observed that in general using hyperstates does lead to greater average returns when exploring an unknown environment than if we used simpler exploration techniques or even state-of-the-art techniques such as the algorithm presented in [Wang *et al.*, 2005]. The drawback of this technique is that it may leave certain areas of the environment unexplored, but this could be the desired behavior. If you consider a physical agent (such as a robot) placed with the task of exploring a new environment, you would want the agent to avoid areas which could cause harm to its mechanical parts or even to other people around it, yet still explore sufficiently to try to reach states with high reward. A situation like that would justify the increase in computation. Furthermore, the method of sparse sampling could allow the hypertree to have a greater depth, and thus, the agent could *see* states that other myopic methods may not be able to. This ability to value actions not only on their immediate but long-term effect makes Bayesian learning a good candidate for exploratory agents.

The next step is to obtain theoretical guarantees for our algorithm. We would also like to implement the algorithms presented in [Strehl *et al.*, 2006] and possibly [Duff, 2002] to empirically compare our algorithm against more of the latest advances in Bayesian decision theory. It would also be interesting to try our algorithm against problems with continuous state and/or action spaces. Placing a Gaussian over sampled

points could assist us in approximating the full hypertree better. Further work could be to apply approximation techniques to the full hyper-tree, in a manner similar to what was done in [de Farias and Roy, 2003] and [Hauskrecht and Kveton, 2004] for MDPs with very large state spaces.

5.1 Acknowledgments

This work was supported in part by NSERC and FQRNT.

References

[Bellman and Kalaba, 1959] R. Bellman and R. Kalaba. On adaptive control processes. In *IRE Trans.*, volume 4, pages 1–9, 1959.

[Brafman and Tennenholtz, 2001] R. I. Brafman and M. Tennenholtz. R-MAX — a general polynomial time algorithm for near-optimal reinforcement learning. In *IJCAI*, pages 953–958, 2001.

[de Farias and Roy, 2003] D.P. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.

[de Farias and Roy, 2004] D.P. de Farias and B. Van Roy. On constraint sampling for the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research*, 29(3):462–478, 2004.

[Dearden *et al.*, 1999] R. Dearden, N. Friedman, and D. Andre. Model-based bayesian exploration. In *Proc. 15th UAI Conference*, pages 150–159, 1999.

[Duff, 2002] M. O. Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachussets, 2002.

[Duff, 2003] M. O. Duff. Design for an optimal probe. In *International Conference on Machine Learning*, 2003.

[Gittins and Jones, 1979] J.C. Gittins and D. Jones. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society, Series B*, 41(2):148–177, 1979.

[Hauskrecht and Kveton, 2004] M. Hauskrecht and B. Kveton. Linear program approximations for factored continuous-state markov decision processes. In *Advances in Neural Information Processing Systems 17*, 2004.

[Kearns and Singh, 1998] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In *Proc. 15th ICML*, pages 260–268, 1998.

[Kearns *et al.*, 1999] M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *IJCAI*, pages 1324–1231, 1999.

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

[Meuleau and Bourgine, 1999] N. Meuleau and P. Bourgine. Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning*, 35(2):117–154, 1999.

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

[Schweitzer and Seidmann, 1985] P. Schweitzer and A. Seidmann. Generalized polynomial approximations in markovian decision processes. *J. Math. Anal. Appl.*, 110:568–582, 1985.

[Strehl and Littman, 2005] A. L. Strehl and M. L. Littman. A theoretical analysis of model-based interval estimation. In *Proc. 21st ICML*, 2005.

[Strehl *et al.*, 2006] A. L. Strehl, L. Li, and M. L. Littman. Incremental model-based learners with formal learning-time guarantees. In *Proc. 21st UAI Conference*, 2006.

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

[Thrun, 1992] S. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, School of Computer Science, 1992.

[Wang *et al.*, 2005] T. Wang, D. Lizotte, M. Bowling, and D. Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *Proceedings of Twenty-First International Conference on Machine Learning*, 2005.