# Approaching Bayes-optimalilty using Monte-Carlo tree search

**John Asmuth**
Rutgers University Computer Science
Piscataway, NJ
jasmuth@cs.rutgers.edu

**Michael Littman**
Rutgers University Computer Science
Piscataway, NJ
mlittman@cs.rutgers.edu

### Abstract

Bayes-optimal behavior, while well-defined, is often difficult to achieve. Recent advances in the use of Monte-Carlo tree search (MCTS) have shown that it is possible to act near-optimally in Markov Decision Processes (MDPs) with very large or infinite state spaces. Bayes-optimal behavior in an unknown MDP is equivalent to optimal behavior in the known belief-space MDP, although the size of this belief-space MDP grows exponentially with the amount of history retained, and is potentially infinite. We show how an agent can use one particular MCTS algorithm, Forward Search Sparse Sampling (FSSS), in an efficient way to approach Bayes-optimality.

## Introduction

In reinforcement learning (RL), a central issue is the exploration/exploitation tradeoff (Sutton & Barto, 1998). Simply put, this dilemma refers to the balance between acting optimally according to the information that you have (exploitation) and acting potentially sub-optimally so as to improve the quality of your information (exploration).

The classical approach to this issue considers the "general" model, and makes its guarantees (if any exist) about an algorithm's behavior on any possible model that satisfies some constraints. A maximum likelihood estimate (MLE) is used with a promise that, if the correct exploration steps are taken, the resulting model is close to the truth.

This frequentist approach to optimal behavior is effective in many scenarios, and has the added benefit that the MLE is often easy to compute. However, providing guarantees for a "general" MDP can cause *over-exploration* beyond what is needed to ensure optimal or near-optimal behavior.

An effective approach to limiting over-exploration is to constrain the model to a class that is easier to learn. For example, if we know that the model dynamics can be treated as a separate problem for any one of a number of state features, we can more efficiently do factor learning (Strehl et al., 2007). Or, if we know that certain groups of states have identical dynamics, we can learn the group dynamics by using relocatable action models (Leffler et al., 2007). The downside to this strategy is that, in most cases, an entirely new algorithm must be invented to take advantage of new constraints.



The Bayesian approach to model estimation in RL (Wilson et al., 2007) introduces the use of model priors, and much of the innovation required moves from algorithm design to prior engineering and inference. While neither of these new issues are trivial, they extend beyond reinforcement learning and general Bayesian techniques from outside of the field will apply, broadening the palette of possible techniques that can be brought to bear.

One effect of introducing Bayesian ideas to reinforcement learning is the concept of *Bayes-optimal* behavior, which chooses actions that maximize expected reward as a function of belief-state. If an agent achieves Bayes-optimal behavior, it circumvents the exploration/exploitation dilemma; information gathering options are built into the definition. As a result, two major aspects of an RL algorithm, *learning* and *planning*, are unified and become simply *planning*.

Unfortunately, the general problem of computing Bayes-optimal behavior is intractable. We will show how an existing planning algorithm, **FSSS**, can be modified to pursue the goal of Bayes-optimal behavior for a very general class of models and model priors.

## Background

Reinforcement learning is a framework for sequential decision-making problems where the dynamics of the environment are not known in advance. In RL, an agent is supplied with an *observation*, to which it responds with an *action*. In response to every action taken, the environment returns a new observation as well as a numerical *reward* signal. The agent's goal is to maximize the total sum of all rewards over time.

Formally, an environment is described as a Markov Decision Process (Puterman, 1994), or MDP. An MDP is the tuple $M = \langle S, A, T, R, \gamma \rangle$, where $S$ is the set of possible states, $A$ is the set of available actions, $T : S \times A \to \Pi(S)$[1] is the transition function, $R : S \times A \to \Pi(\Re)$ is the reward function, and $\gamma$ is a discount factor.

A policy $\pi : S \to A$ describes what action an agent takes in a given state. The optimal policy for some MDP $M$ is

[1] Here, we use the notation that $\Pi(X)$ is the set of probability distributions over the set $X$.

defined as follows:

$$\begin{aligned} \pi^* &= \operatorname*{argmax}_a Q(s,a) \\ Q(s,a) &= E[R(s,a)] + \gamma E_{s' \sim T(s,a)}[V(s')] \\ V(s) &= \max_a Q(s,a). \end{aligned}$$

For any given MDP prior $\phi(M)$, a corresponding belief-MDP $m_\phi$ can be constructed by considering the set of all possible histories $H$, which consists of every finite sequence of steps in the environment. The state space of the belief-MDP $\mathcal{S} = S \times H$ pairs "real" states with histories. The action space $A$ remains unchanged. Since the belief-state includes history, next states in the transition function are constrained to belief-states that extend the previous belief-state with the last occurring transition and reward. Because states are completely observable, histories can be summarized by counts of individual state transitions. It is easiest to express the new transition and reward functions as a single joint distribution $T\text{-}R_\phi(\langle s,h\rangle, a) = \Pi(\langle s', h \cup (s,a,s',r)\rangle, r)$.

When $T\text{-}R_\phi$ is constructed in such a way that the likelihood $P(\langle s', h \cup (s,a,s',r)\rangle, r | \langle s,h\rangle, a) = \int_M P(s',r|s,a,M)\phi(M|h)dM$, the optimal policy in $m_{\phi|h}$ corresponds to the Bayes-optimal policy, given the MDP prior $\phi$. In many cases, transition likelihoods are easy to compute or sample from. For example, the **Flat-Dirichlet-Multinomial** (Poupart et al., 2006), or **FDM**,

$$\begin{aligned} \theta_{s,a} &\sim Dir(\alpha) \\ T(s,a) &\sim Mult(\theta_{s,a}), \end{aligned}$$

holds that the next-state distributions for each state-action pair are i.i.d. with a Dirichlet prior. In this case, $P(s'|s,a,h) \propto C_{s,a}(s') + \alpha_{s'}$, where $C_{s,a}(s')$ is the number of times a transition to $s'$ has been observed when action $a$ was taken in state $s$, and $\alpha_{s'}$ is the element in the hyper-parameter vector $\alpha$ corresponding to $s'$. Note that a contribution of the current paper is to handle general distributions beyond Dirichlet priors.

The obvious intractability problem comes from the size of the belief-MDP's state space $\mathcal{S}$, which grows exponentially with the length of the amount of history to be considered. Even for a very small MDP, the corresponding belief-MDP quickly becomes unmanageable. The use of many standard planning techniques, such as value iteration or policy iteration (Sutton & Barto, 1998; Puterman, 1994), becomes impossible with an infinite horizon domain.

*Exact Bayes-optimal* behavior is impossible to compute efficintly for general MDPs and priors (including **FDM**), so we must rely upon approximations. *Approximate Bayes-optimal* behavior, in which the value of each action selected is within $\epsilon$ of the value of the exact Bayes-optimal action, is easier to achieve than exact Bayes-optimal behavior, but is still intractable. *Near Bayes-optimal* behavior, which is approximately Bayes-optimal for all but a polynomial number of steps, can be efficiently achieved in many situations and is the focus of our work.

## Related Work

The approach we propose in this paper has roots in a number of existing algorithms, which we sketch next.

**Bayesian Exploration/Exploitation Trade-off in LEarning** (Poupart et al., 2006), or **BEETLE**, is an algorithm that uses the belief-MDP formulation of Bayesian RL in order to achieve approximately Bayes-optimal behavior. It uses the **FDM** prior and known properties of the value function to calculate an approximation over all states.

**Bayesian Exploration Bonus** (Kolter & Ng, 2009), or **BEB**, is a near Bayes-optimal algorithm for the **FDM** prior and known rewards. It acts greedily according to the maximum likelihood estimation of the MDP, but uses the alternate reward function $R_{\text{BEB}}(s,a) = R(s,a) + \beta/(1 + n(s,a))$, where $\beta$ is a domain-dependent constant and $n(s,a)$ is the number of times action $a$ has been performed in state $s$. This reward-supplement strategy is also used in PAC-MDP (Kakade, 2003) approaches (Strehl & Littman, 2008), and biases an agent toward states in which it has less experience. The particular bonus used by **BEB** will cause the agent to explore enough to be near Bayes-optimal without the over-exploring seen in PAC-MDP algorithms. There are ways to use posterior variance to create reward bonuses for more general Bayesian priors (Sorg et al., 2010).

**Bayesian Dynamic Programming** (Strens, 2000), or **Bayesian-DP**, and **Best Of Sampled Set** (Asmuth et al., 2009), or **BOSS**, are two examples of Bayesian RL algorithms that can work with a much more flexible prior. They both use samples from the MDP posterior. **Bayesian-DP** will sample an MDP at the beginning of the experiment, and will resample the model when the current one has been in use for some threshold number of steps. It acts greedily according to the most recent sample. **BOSS** will sample $C$ models every time a particular state-action pair has been tried $B$ times, and then combine all $C$ models in such a way as to have each state's value be at least as great as that state's value in each of the sampled MDPs. Both algorithms rely on uncertainty in the posterior causing variance in the samples: variance in the samples causes optimistic value estimates, which in turn drive the agent to visit under-explored parts of the state space.

**Sparse Sampling** (Kearns et al., 1999) works by recursively expanding a full search tree up to a certain depth $d$. At the root, each of the $A$ actions is chosen a constant number of times $C$, yielding a set of $A \cdot C$ children. Sparse sampling is then run on each of the children with a recursion depth one less than the root's. Once the tree is fully created, the leaves are each assigned a value of zero. Then, starting at the leaves, the values are backed up and combined via the Bellman equation, giving the parents' values, until the root's value is determined. The total number of nodes in this search tree is $(AC)^d$, making the algorithm impractical to run in all but the most trivial of domains.

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

2006; Walsh et al., 2010; Wang et al., 2005).

**Bayesian Sparse Sampling** (Wang et al., 2005) is a modification of **Sparse Sampling** that applies only in the Bayesian setting. Instead of a full tree expansion, **Bayesian Sparse Sampling** preferentially expands only promising parts of the tree by performing rollouts, or simulated trajectories, up to the specified depth. On a given rollout, the action for a particular node's belief-state is chosen by sampling a model from the posterior and solving this model exactly for the current "real" state. This action-selection strategy is myopic, but the algorithm can still achieve Bayes-optimal behavior in the limit because the method for computing the resulting policy is the same as in **Sparse Sampling**; it propagates values from the leaves back towards the root. This algorithm is limited by the exact-solve step within the inner loop, but also works in domains with continuous action spaces, unlike **Sparse Sampling**.

**Upper Confidence bounds on Trees** (Kocsis & Szepesvári, 2006), or **UCT**, is a recent Monte-Carlo tree search idea that has gained notice through good performance in computer Go (Gelly & Silver, 2008). Like **Bayesian Sparse Sampling**, **UCT** preferentially expands the search tree by performing rollouts. In addition to the value estimates for each node, which are computed by running backups backwards along trajectories, **UCT** maintains upper confidence bounds using the **UCB** algorithm (Auer et al., 2002). This algorithm is very aggressive and typically will under-explore.

**Forward Search Sparse Sampling** (Walsh et al., 2010), or **FSSS**, is the approach we adopt in this paper. It also preferentially expands the search tree through the use of rollouts. It is outlined in Algorithm 1. Unlike either **Bayesian Sparse Sampling** or **UCT**, it retains the attractive guarantees of the original **Sparse Sampling** algorithm. **FSSS** maintains hard upper and lower bounds on the values for each state and action so as to direct the rollouts; actions are chosen greedily according to the upper bound on the value, and the next state is chosen such that it is the most uncertain of the available candidates (according to the difference in its upper and lower bounds).

**FSSS** will find the action to take from a given state $s_0$, which will be the root of the search tree. The tree is expanded by running $t$ trajectories, or rollouts, of length $d$. There are theoretically justified ways to choose $t$ and $d$, but in practical applications they are knobs used to balance computational overhead and accuracy. To run a single rollout, the agent will call Algorithm 2, FSSS-Rollout$(s_0, d, 0)$. The values $U_d(s)$ and $L_d(s)$ are the upper and lower bounds on the value of the node for state $s$ at depth $d$, respectively. Each time a rollout is performed, the tree will be expanded. After at most $(AC)^d$ rollouts are finished (but often less in practice), **FSSS** will have expanded the tree as much as is possibly useful, and will agree with the action chosen by **Sparse Sampling**.

**Input**: state $s$, max depth $d$, #trajectories $t$
**Output**: estimated value for state $s$
**for** $t$ *times* **do**
  FSSS-Rollout$(s, d, 0)$
$\hat{V}(s) \leftarrow \max_a U_d(s, a)$
**return** $\hat{V}(s)$

**Algorithm 1:** FSSS$(s, d, t)$

**Input**: state $s$, max depth $d$, current depth $l$
**if** *Terminal*$(s)$ **then**
  $U_d(s) = L_d(s) = 0$
  **return**
**if** $d = l$ **then**
  **return**
**if** $\neg$*Visited*$_d(s)$ **then**
  Visited$_d(s) \leftarrow$ true
  **foreach** $a \in A$ **do**
    $R_d(s, a), \text{Count}_d(s, a, s'), \text{Children}_d(s, a) \leftarrow 0, 0, \{\}$
    **for** $C$ *times* **do**
      $s', r \sim T(s, a), R(s, a)$
      $\text{Count}_d(s, a, s') \leftarrow \text{Count}_d(s, a, s') + 1$
      $\text{Children}_d(s, a) \leftarrow \text{Children}_d(s, a) \cup \{s'\}$
      $R_d(s, a) \leftarrow R_d(s, a) + r/C$
      **if** $\neg$*Visited*$_{d+1}(s')$ **then**
        $U_{d+1}(s'), L_{d+1}(s') = V_{\max}, V_{\min}$
  Bellman-backup$(s, d)$
$a \leftarrow \text{argmax}_a U_d(s, a)$
$s' \leftarrow \text{argmax}_{s'}(U_{d+1}(s') - L_{d+1}(s')) \cdot \text{Count}_d(s, a, s')$
FSSS-Rollout$(s', d, l + 1)$
Bellman-backup$(s, d)$
**return**

**Algorithm 2:** FSSS-Rollout$(s, d, l)$

## Bayesian FSSS

This paper's contribution, **Bayesian FSSS** (**BFS3**) is the application of **FSSS** to a belief-MDP and is outlined in Algorithm 3. For some MDP prior $\phi(M)$, the joint transition and reward function $T$-$R_\phi$ is constructed such that

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

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

$$M \sim \phi|h \quad (1)$$

$$s', r \sim T_M(s, a), R_M(s, a). \quad (2)$$

First, an MDP $M$ is sampled from the posterior $\phi|h$, and then the next state and reward are sampled from $M$. To reconstruct the resulting belief-state, we pack $s'$ with the new history made by augmenting $h$ with $(s, a, s', r)$, resulting in $\langle s', h \cup (s, a, s', r)\rangle$.

Figure 1 shows how an agent can step through the search tree while making observations. The grayed trail represents the agent's history. The history for any particular node is the path from the root to that node.

In many cases, the history $h$ can be summarized by a more compact sufficient statistic. The sufficient statistic for **FDM** (and for any discrete state and discrete action MDP) is the set of histograms indicating how many times each state $s'$

**Input**: state $s$, history $h$, depth $d$, #trajectories $t$
**Output**: action to take in state $s$
**if** $\langle s, h\rangle \in$ *solved-belief-states* **then**
  **return** $\pi(\langle s, h\rangle)$
**foreach** $a \in A$ **do**
  **for** $C$ *times* **do**
    $\langle s', h'\rangle, r \sim T\text{-}R_\phi(\langle s, h\rangle, a)$
    $q(a) \leftarrow q(a) + \frac{1}{C}\left[r + \gamma FSSS(\langle s', h'\rangle, d, t)\right]$
solved-belief-states $\leftarrow$ solved-belief-states $\bigcup\{\langle s, h\rangle\}$
$\pi(\langle s, h\rangle) \leftarrow \text{argmax}_a\, q(a)$
**return** $\pi(\langle s, h\rangle)$

**Algorithm 3:** BFS3$(s, h, d, t)$

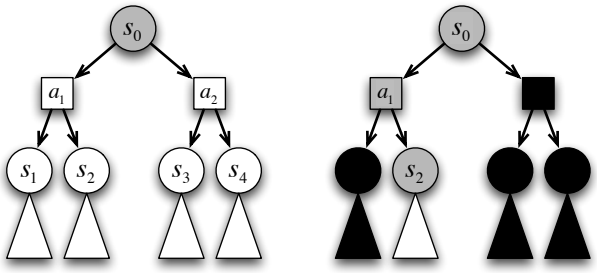


Figure 1: **Left:** the search tree when the agent is at state $s_0$ with no further observations. **Right:** the agent has taken a single step in the world by performing action $a_1$, resulting in a transition to state $s_2$. White nodes indicate hypotheticals that may or may not happen. Grayed nodes indicate things that have been observed. Blackened nodes indicate data that is no longer pertinent and can be thrown out.

has resulted from a particular state-action pair. Due to conjugacy, next-states can be sampled using **FDM** without sampling an intermediate MDP. The next-state distribution is a multinomial with weights proportional to the corresponding histogram added to the prior vector $\alpha$.

## Experimentation

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

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

**BFS3**, with the right prior, can handle unknown rewards. In many domains, there are only a few possible reward values. For example, many path-finding domains give a reward

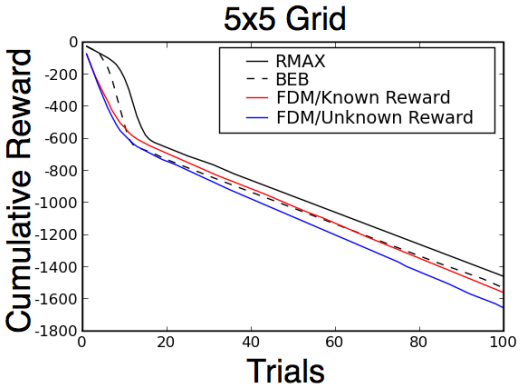


Figure 2: Here **BFS3/FDM**, with both known rewards and unknown rewards with a DP prior, is compared against the known-reward **BEB**, and **RMAX** (Brafman & Tennenholtz, 2002), a well known algorithm with PAC-MDP guarantees, with $M = 5$. Results are averaged over 40 runs.

of $-1$ for all actions. Or, there is a reward for a particular outcome that can be achieved from multiple states: these states would share the same reward value. To represent this common structure in a generative model, the Dirichlet Process (DP) (MacEachern & Muller, 1998) may be used:

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

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

In Figure 2, **BFS3**, with the unknown reward prior, is shown to suffer no significant performance penalty compared to **BFS3** with the known-reward prior and to **BEB** (which also uses the known-reward prior). The domain is a $5 \times 5$ grid world, where the agent must find its way from one corner to the other. The agent can choose to go north, east, south or west, and with probability $0.2$ it will go in a direction perpendicular to the one intended.

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

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

The Paint/Polish world (Walsh et al., 2009) provides a situation where the simple and convenient **FDM** prior is insufficient. The size of the state-space grows exponentially with the number of cans to paint (each of which introduces four binary features). Figure 3 shows the results with a single can (and $2^4$ states) and the results with two cans (and $2^{4^2}$ states). If the number of cans is increased to four, **FDM** cannot find a good policy in a reasonable amount of time while **Factored-Object** can do so in around ten trials.

**BFS3** can also be used to apply Bayesian modeling to POMDPs. Wumpus World (Russell & Norvig, 1994) is based a classic computer game in which an agent wanders through a $4 \times 4$ maze filled with fog, making it impossible to see past its current cell. Even though the agent cannot see, it can feel a breeze if there is a pit in an adjacent cell, and smell a stench if there is a Wumpus[2] near-by. If the agent falls into a pit, it will remain there forever. If the agent runs into the Wumpus, it is eaten. If the agent shoots its one arrow in the direction of the Wumpus, the Wumpus is slain. If the arrow misses the Wumpus, the trial ends and presumably the agent goes home. We replicate the dynamics presented by Sorg et al. (2010).

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

We ran **BFS3** on Wumpus World with a search depth of 15, and varied the number of trajectories per step. Agents with 500, 1000, and 10000 trajectories per step averaged 0.629, 0.682 and 0.695 cumulative reward, respectively. Averages were taken over 1000 attempts. We compare this to a variance-based reward bonus strategy (Sorg et al., 2010) which, when tuned, averaged 0.508.

That **BFS3** performs better in Wumpus World as the computation budget is increased supports our argument that the algorithm has a *computational resources* knob which, when tuned higher and higher, causes the agent's behavior to get closer and closer to being Bayes-optimal.

## Concluding Remarks

The use of Bayesian methods with flexible priors has been a boon to machine learning and artificial intelligence. Similarly, the use of model priors in Bayesian reinforcement learning is a powerful tool for incorporating prior knowledge into an algorithm in a principled way. However, most algorithms that make use of any prior use only the Flat Dirichlet Multinomial (**FDM**). While appropriate for many situations, this prior does not provide for any generalization across states or for continuous state spaces severely limiting its applicability.

**Cluster** is an interesting prior that can be used instead of **FDM**. In degenerate cases, it will converge to the same point as **FDM**, but very often it will detect that groups of states share dynamics and can be clustered together. Algorithms that limit themselves to **FDM** cannot make use of this or any other generalizing prior.

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

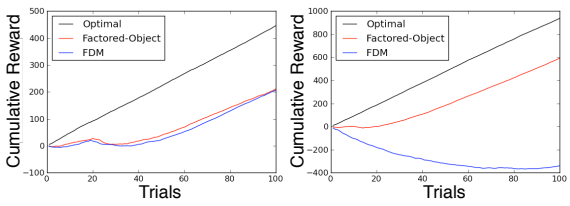


Figure 3: **BFS3** with **FDM** and **Factored-Object** priors. **Left:** Paint/Polish with 1 can. **FDM** and **Factored-Object** are identical for one object, and give the same performance. **Right:** Paint/Polish with 2 cans. **Factored-Object** outperforms **FDM** because it is better able to generalize. Results are averaged over 40 runs.

Though limited in other ways, **FDM** does have at least one property that makes it more attractive than many other prior distributions: it makes posterior sampling trivial. Especially for MCTS approaches, posterior sampling needs to be fast, as it is done often. The aforementioned **Cluster** prior is difficult to use with MCTS strategies because sampling from its posterior involves Markov chain Monte-Carlo techniques (Neal, 2000), and requires a great deal of computation.

**BOSS** (Asmuth et al., 2009) can make use of flexible priors, and uses sample redundancy and posterior variance to ensure constant optimism, and is PAC-MDP. However, it is very conservative and will often over-explore. Like **BOSS**, the variance-based reward bonus approach (Sorg et al., 2010) draws upon posterior variance of any prior to encourage exploration to unknown states, but does so in a different way that gives it near Bayes-optimality.

The algorithm we have introduced, **BFS3**, has the best of both worlds: it can make use of a wide variety of priors, and it is near Bayes-optimal. Its weakness is computation. For some prior distributions (**Cluster**), posterior samples can be expensive, and for others a large branching factor $C$ must be used. Both increase the computational power required to ensure good behavior. Despite this weakness the great flexibility it offers, in terms of domains and of priors, make it an attractive application of Bayesian techniques for reinforcement learning.

PAC-MDP and near Bayes-optimal differ in important ways. Near Bayes-optimal is a claim made in the context of some prior distribution, where PAC-MDP is a claim made in the context of any possible underlying model. Algorithms that are PAC-MDP will therefore over-explore, while near Bayes-optimal algorithms will explore just the right amount, according to the prior.

The exploration/exploitation dilemma is central to the RL community. PAC-MDP and Bayes-optimal guarantees represent two important approaches to this problem. PAC-MDP algorithms address the issue by spending a budget of exploration steps toward improving their models to the point

at which they are accurate. Bayes-optimal algorithms address the issue by planning in belief-space, turning the exploration/exploitation dilemma into just exploitation: learning is planning.

## References


Asmuth, J., Li, L., Littman, M., Nouri, A., & Wingate, D. (2009). A Bayesian sampling approach to exploration in reinforcement learning. *Proceedings of the 25th Conference on Uncertainty in Artifical Intelligence (UAI-09)*.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*, 235–256.

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX—A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, *3*, 213–231.

Gelly, S., & Silver, D. (2008). Achieving master level play in 9x9 computer go. *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3* (pp. 1537–1540). Chicago, Illinois: AAAI Press.

Kakade, S. M. (2003). *On the sample complexity of reinforcement learning*. Doctoral dissertation, Gatsby Computational Neuroscience Unit, University College London.

Kearns, M., Mansour, Y., & Ng, A. Y. (1999). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)* (pp. 1324–1331).

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. *Proceedings of the 17th European Conference on Machine Learning (ECML-06)* (pp. 282–293). Springer Berlin / Heidelberg.

Kolter, J. Z., & Ng, A. Y. (2009). Near-Bayesian exploration in polynomial time. *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 513–520). New York, NY, USA: ACM.

Leffler, B. R., Littman, M. L., & Edmunds, T. (2007). Efficient reinforcement learning with relocatable action models. *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*.

MacEachern, S. N., & Muller, P. (1998). Estimating mixture of Dirichlet process models. *Journal of Computational and Graphical Statistics*, *7*, 223–238.

Neal, R. M. (2000). Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, *Vol. 9*, pp. 249–265.

Poupart, P., Vlassis, N., Hoey, J., & Regan, K. (2006). An analytic solution to discrete Bayesian reinforcement learning. *Proceedings of the 23rd International Conference on Machine Learning* (pp. 697–704).

Puterman, M. L. (1994). *Markov decision processes—discrete stochastic dynamic programming*. New York, NY: John Wiley & Sons, Inc.

Russell, S. J., & Norvig, P. (1994). *Artificial intelligence: A modern approach*. Englewood Cliffs, NJ: Prentice-Hall.

Sorg, J., Singh, S., & Lewis, R. L. (2010). Variance-based rewards for approximate Bayesian reinforcement learning. *Proceedings of the 26th Conference on Uncertainty in Artifical Intelligence (UAI-10)*.

Strehl, A. L., Diuk, C., & Littman, M. L. (2007). Efficient structure learning in factored-state MDPs. *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI-07)*.

Strehl, A. L., & Littman, M. L. (2008). An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, *74*, 1309–1331. Special Issue on Learning Theory.

Strens, M. J. A. (2000). A Bayesian framework for reinforcement learning. *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)* (pp. 943–950).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. The MIT Press.

Walsh, T., Goschin, S., & Littman, M. (2010). Integrating sample-based planning and model-based reinforcement learning. *Proceedings of the Association for the Advancement of Artificial Intelligence*.

Walsh, T. J., Szita, I., Diuk, C., & Littman, M. L. (2009). Exploring compact reinforcement-learning representations with linear regression. *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence* (pp. 591–598). Arlington, Virginia, United States: AUAI Press.

Wang, T., Lizotte, D., Bowling, M., & Schuurmans, D. (2005). Bayesian sparse sampling for on-line reward optimization. *ICML '05: Proceedings of the 22nd International Conference on Machine Learning* (pp. 956–963). New York, NY, USA: ACM.

Wilson, A., Fern, A., Ray, S., & Tadepalli, P. (2007). Multi-task reinforcement learning: A hierarchical Bayesian approach. *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007)* (pp. 1015–1022).