# Model-based Bayesian Reinforcement Learning in Partially Observable Domains

**Pascal Poupart**
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1
ppoupart@cs.uwaterloo.ca

**Nikos Vlassis**
Dept. of Production Engineering & Management
Technical University of Crete
Crete, Greece
vlassis@dpem.tuc.gr

**Abstract**

Bayesian reinforcement learning in partially observable domains is notoriously difficult, in part due to the unknown form of the beliefs and the optimal value function. We show that beliefs represented by mixtures of products of Dirichlet distributions are closed under belief updates for factored domains. Belief monitoring algorithms that use this mixture representation are proposed. We also show that the optimal value function is a linear combination of products of Dirichlets for factored domains. Finally, we extend BEETLE, which is a point-based value iteration algorithm for Bayesian RL in fully observable domains, to partially observable domains.

## 1 Introduction

Reinforcement learning (RL) is a popular framework to tackle sequential decision making problems when the dynamics of the environment are unknown. A wide range of tasks can be formulated as RL problems including game playing (Tesauro 1995), robotic locomotion (Hohl & Stone 2004), helicopter control (Ng *et al.* 2003), etc.

When the sensors of the decision maker provide only partial and/or noisy information about the state of the environment, RL becomes notoriously difficult. As a result, many approaches to partially observable reinforcement learning (PORL) do not attempt to estimate the state of the environment and generally avoid to explicitly model the environment (e.g., policy gradient algorithms (Meuleau *et al.* 1999; Ng, Parr, & Koller 2000; Aberdeen & Baxter 2002)). While such *model-free* approaches are relatively simple, they cannot take advantage of prior knowledge about the environment to reduce the amount of training. In contrast, approaches that explicitly model the environment can facilitate the encoding of prior knowledge about the model, but at the cost of a significant increase in computational complexity. In fact, to simplify things, many *model-based* approaches decouple model learning from policy optimization by first learning a model in controlled experiments (e.g., using EM) and then using planning techniques (e.g., POMDP algorithms) to optimize the course of action. An exception is the recent work by Jaulmes, Pineau et al. (2005) on active learning in POMDPs.

Building on our previous work in Bayesian RL for fully observable domains (2006), we develop a Bayesian model-based approach for PORL in discrete factored domains. We show that Bayesian model learning yields a mixture of products of Dirichlets over the model parameters. We also show that the optimal value function in partially observable domains has an analytic form parameterized by a set of linear combinations of products of Dirichlets. We then adapt our BEETLE algorithm (Poupart *et al.* 2006) to PORL. Finally, we demonstrate that our Bayesian PORL approach implicitely performs active learning in a way that is more general than the technique by Jaulmes, Pineau et al. (2005) since it isn't restricted to oracles that reveal the underlying state.

## 2 Partially observable Markov decision processes (POMDPs)

We begin by establishing the notation used throughout the paper. Upper case letters (i.e., $X$) denote variables, lower case letters (i.e., $x$) denote values and bold letters (i.e., $\mathbf{X}$) denote sets.

Reinforcement learning (RL) is the process by which an agent optimizes its course of action given some feedback from the environment. In partially observable domains, RL can be formalized by a partially observable Markov decision process (POMDP) that we define by a dynamic decision network (DDN) $G = \langle \mathbf{X}, \mathbf{X}', E \rangle$ over two time slices (see Figure 1). Let $\mathbf{X}$ and $\mathbf{X}'$ be the sets of nodes (i.e., variables) in two consecutive time slices and $E$ be the set of edges indicating the parents of each variable in $\mathbf{X}'$. The parents of each node are always in the same or previous slice to satisfy the Markov property. Let $\mathbf{S} \subseteq \mathbf{X}$ be the subset of nodes that are random *state* variables encoding features of the environment. Let $\mathbf{O} \subseteq \mathbf{S}$ be the subset of *observable* variables corresponding to sensor measurements. In fully observable domains $\mathbf{S} = \mathbf{O}$, but in partially observable domains $\mathbf{O} \subseteq \mathbf{S}$. Let $\mathbf{R} \subseteq \mathbf{S}$ be the subset of *reward* variables indicating the quantities that must be maximized. While the reward variables are utility nodes, we treat them as random variables and view them as a subset of the

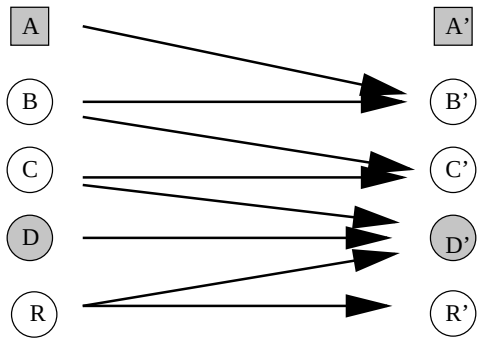


Figure 1: Dynamic decision network; set of all variables: $\mathbf{X} = \{A, B, C, D, R\}$; set of state variables: $\mathbf{S} = \{B, C, D, R\}$; observable variables are shaded: $\mathbf{O} = \{D\}$; action variables are denoted by squares: $\mathbf{A} = \{A\}$; reward variables: $\mathbf{R} = \{R\}$.

state variables. As we will see shortly, this will be convenient when learning the reward function. Note also that $\mathbf{R}$ is not necessarily a subset of $\mathbf{O}$ since rewards are not generally quantities directly measurable by the sensors. Let $\mathbf{A} \subseteq \mathbf{X}$ be the subset of *action* variables which are decision nodes. The conditional probability distributions (CPDs) $\Pr(S'|\mathbf{PA}_{S'})$ of each state variable $S' \in \mathbf{S}'$ encode the dynamics of the model. Here $\mathbf{PA}_X$ denotes the set of parent variables of $X$ and $\mathbf{pa}_X$ denotes a joint assignment of values to the set of parent variables of $X$. Since the observation and reward variables are subsets of the state variables, the CPDs of the state variables jointly encode the transition, observation and reward models. The action variables do not have CPDs since they are decision variables set by the agent. As we will see shortly, encoding POMDPs as DDNs offers three advantages: (i) learning the transition, observation and reward models can be done with a single approach that learns CPDs in general, (ii) prior knowledge can be encoded in the graphical structure to reduce the number of CPD parameters that must be learned and (iii) DDNs are closed under Bayesian learning of the CPD parameters.

The problem of POMDP planning consists of finding a good policy by controlling the action variables assuming that $\mathbf{X}$, $\mathbf{S}$, $\mathbf{R}$, $\mathbf{O}$, $\mathbf{A}$ are known, the domain of each variable is known, the edges $E$ are known and the CPDs are known. A policy consists of a mapping from histories of observable quantities to actions. Since observations and actions are the only observable quantities, histories consist of sequences of past actions and observations. However, histories grow over time, so in practice it is common to consider policies that are mappings from *belief distributions* to actions. A belief distribution $b(\mathbf{S}) = \Pr(\mathbf{S})$ is a fixed-length sufficient statistic of histories that correspond to the probability distribution over the state variables at the current time step. Given a belief $b(\mathbf{S})$, we can compute an updated belief $b_{\mathbf{a},\mathbf{o}'}(\mathbf{S}')$ after executing $\mathbf{a}$ and observing $\mathbf{o}'$ using Bayes' theorem:

$$b_{\mathbf{a},\mathbf{o}'}(\mathbf{s}') = k\delta([\mathbf{s}']_{\mathbf{O}'} = \mathbf{o}') \sum_{\mathbf{s}} b(\mathbf{s}) \Pr(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \quad (1)$$

Here $k$ is a normalization constant, $[\mathbf{s}']_{\mathbf{O}'}$ is the subset of state values corresponding to the observable variables $\mathbf{O}'$ and $\delta(p)$ is a kronecker delta that returns 1 when $p$ is true and 0 otherwise.

The goal is to find a policy $\pi : B \rightarrow dom(\mathbf{A})$ that maximizes the expected total return. More precisely, the value $V^\pi$ of a policy $\pi$ starting with belief $b$ is measured by the sum of future discounted rewards:

$$V^\pi(b) = \sum_t \gamma^t E_\pi[E_{b_t}[\mathbf{R}]] \quad (2)$$

Here, $\gamma$ is a discount factor in $[0, 1)$, $E_{b_t}[\mathbf{R}] = \sum_{R \in \mathbf{R}} \sum_{r \in dom(R)} b_t(r) r$ is the immediate expected reward according to belief $b_t$, and $E_\pi[\cdot] = \sum_{b_t} \Pr(b_t|\pi)\cdot$ is the expected value earned at time step $t$ under policy $\pi$. A policy $\pi^*$ is optimal when its value function $V^*$ is at least as high as any other policy for all beliefs (i.e., $V^*(b) \geq V^\pi(b)\ \forall \pi, b$). The optimal value function also satisfies Bellman's equation:

$$V^*(b) = \max_{\mathbf{a}} E_b[\mathbf{R}] + \gamma \sum_{\mathbf{o}'} \Pr(\mathbf{o}'|b, \mathbf{a}) V(b_{\mathbf{a},\mathbf{o}'}) \quad (3)$$

where $\Pr(\mathbf{o}'|b, \mathbf{a}) = \sum_{\mathbf{s}} b(\mathbf{s}) \Pr(\mathbf{o}'|\mathbf{s}, \mathbf{a})$. Value iteration algorithms optimize the value function by iteratively computing the right hand side of Bellman's equation. Alternatively, policy search algorithms directly optimize a policy.

## 3 Partially observable reinforcement learning (PORL)

In many real-world problems, the dynamics of the environment encoded by the CPDs may not be known. We consider the more general problem of PORL in which we seek to optimize a policy given that $\mathbf{X}$, $\mathbf{S}$, $\mathbf{R}$, $\mathbf{O}$, $\mathbf{A}$ are known, their domain is known and finite, the edges $E$ are known, but the CPDs are (partially or completely) unknown.[1] We develop a Bayesian model-based approach to PORL. Section 3.1 describes how to learn the model in a Bayesian way, while Section 3.2 characterizes the optimal value function.

### 3.1 Bayesian model learning

As the agent interacts with the environment, it gathers observations that can be used to construct a model. In Bayesian learning, this process is achieved by modeling each unknown CPD parameter by a random variable. We start with a prior belief over the model variables, which is subsequently updated after each action-observation pair. More precisely, for each unknown conditional probability $\Pr(s'|\mathbf{pa}_{S'})$, we create a new model

[1] The more general problem of structure learning when the edges $E$ and the state variables $\mathbf{S}$ are unknown is subject to future work.

variable $\Theta_{s'|\mathbf{pa}_{S'}}$ that can take any value $\theta_{s'|\mathbf{pa}_{S'}} = \Pr(s'|\mathbf{pa}_{S'})$. Here $\Theta$ is a random variable and $\theta \in dom(\Theta) = [0,1]$ is any probability value. We also denote by $\mathbf{\Theta}_{S'|\mathbf{pa}_{S'}}$ the set of all $\Theta_{s'|\mathbf{pa}_{S'}}$ random variables for each $s' \in dom(S')$. Similarly $\mathbf{\Theta}_{S'|\mathbf{PA}_{S'}}$ is the set of all $\Theta_{s'|\mathbf{pa}_{S'}}$ for each $(s', \mathbf{pa}_{S'}) \in dom(S') \times dom(\mathbf{PA}_{S'})$ and $\mathbf{\Theta}$ is the set of all $\mathbf{\Theta}_{S'|\mathbf{PA}_{S'}}$ for each $S' \in \mathbf{S}'$. We can then augment the DDN representation of a POMDP with model variables in such a way that each conditional distribution is now conditioned on $\mathbf{\Theta}$ (i.e., $Pr(S'|\mathbf{PA}_{S'}, \mathbf{\Theta}) = \mathbf{\Theta}_{S'|\mathbf{PA}_{S'}}$). Augmenting the DDN in Figure 1 with $\mathbf{\Theta}$ yields the DDN in Figure 2. Assuming that the unknown model is static, the model variables do not change over time (i.e., $\Pr(\mathbf{\Theta}'|\mathbf{\Theta}) = 1$ when $\mathbf{\Theta} = \mathbf{\Theta}'$ and 0 otherwise).[2] It is interesting to note that the CPDs of the augmented DDN in Figure 2 do not have unknown parameters, which allows us to compute the joint belief over $\mathbf{S}$ and $\mathbf{\Theta}$ by inference at each time step. In fact, model learning can be viewed as belief updating based on each action-observation pair at each time step. Similar to Equation 1, we can update joint beliefs as follows:

$$b_{\mathbf{a},\mathbf{o}'}(\mathbf{s}', \theta) = k\delta([\mathbf{s}']_{\mathbf{O}'} = \mathbf{o}') \sum_{\mathbf{s}} b(\mathbf{s}, \theta) \Pr(\mathbf{s}'|\mathbf{s}, \mathbf{a}, \theta) \tag{4}$$

Despite the fact that $\mathbf{\Theta}$ is a set of continuous model variables, belief updates can be done in closed form when starting with a mixture of Dirichlet distributions as the prior (Heckerman 1999). Dirichlet distributions are a class of density functions over the space of discrete probability distributions of the form $\mathcal{D}(\mathbf{\Theta}; \mathbf{n}) = k\Pi_i \Theta_i^{n_i - 1}$. They are parameterized by a set $\mathbf{n}$ of non-negative hyperparameters $n$ such that each $n_i$ can be interpreted as having observed the $\Theta_i$-probability event $n_i - 1$ times. Dirichlets distributions have the nice property that they are *conjugate priors* of multinomial distributions, which means that updating a Dirichlet prior according to a multinomial sample yields a Dirichlet posterior. In our case, observations are samples of mixtures of products of multinomial so mixtures of products of Dirichlets are conjugate priors.

**Theorem 1** *Mixtures of products of Dirichlets are closed under belief updates.*

**Proof:** Let the prior belief be a mixture of products of Dirichlets $b(\mathbf{s}, \theta) = \sum_i c_{i,\mathbf{s}} \prod_{S', \mathbf{pa}_{S'}} \mathcal{D}_{i,\mathbf{s}}(\theta_{S'|\mathbf{pa}_{S'}})$ with coefficients $c_{i,\mathbf{s}}$, then the posterior belief is again a mixture of products of Dirichlets:

$$b_{\mathbf{a},\mathbf{o}'}(\mathbf{s}', \theta) = k\delta([\mathbf{s}']_{\mathbf{O}'} = \mathbf{o}') \sum_{\mathbf{s}} \Pr(\mathbf{s}'|\mathbf{s}, \mathbf{a}, \theta) b(\mathbf{s}, \theta)$$

$$= k\delta([\mathbf{s}']_{\mathbf{O}'} = \mathbf{o}') \sum_{\mathbf{s}} [\prod_{S'} \theta_{[\mathbf{s}'|\mathbf{s},\mathbf{a}]_{S'|\mathbf{PA}_{S'}}}]$$
$$\sum_i c_{i,\mathbf{s}} \prod_{S', \mathbf{pa}_{S'}} \mathcal{D}_{i,\mathbf{s}}(\theta_{S'|\mathbf{pa}_{S'}}; \mathbf{n}_{S'|\mathbf{pa}_{S'}})$$

$$= k\delta([\mathbf{s}']_{\mathbf{O}'} = \mathbf{o}') \sum_{i,\mathbf{s}} c_{i,\mathbf{s}} \prod_{S', \mathbf{pa}_{S'}}$$
$$\mathcal{D}_{i,\mathbf{s}}(\theta_{S'|\mathbf{pa}_{S'}}; \mathbf{n}_{S'|\mathbf{pa}_{S'}} + \delta(S'|\mathbf{pa}_{S'} = [\mathbf{s}'|\mathbf{s}, \mathbf{a}]_{S'|\mathbf{PA}_{S'}}))$$

If we use a new index $j$ for each $(i, \mathbf{s})$-pair, we can define $c_{j,\mathbf{s}'} = k\delta([\mathbf{s}']_{\mathbf{O}'} = \mathbf{o}')c_{i,\mathbf{s}}$ and $\mathcal{D}_{j,\mathbf{s}'}(\theta_{S'|\mathbf{pa}_{S'}}) = \mathcal{D}_{i,\mathbf{s}}(\theta_{S'|\mathbf{pa}_{S'}}; \mathbf{n}_{S'|\mathbf{pa}_{S'}} + \delta(S'|\mathbf{pa}_{S'} = [\mathbf{s}'|\mathbf{s}, \mathbf{a}]_{S'|\mathbf{PA}_{S'}}))$, making it clear that the posterior is also a mixture of products of Dirichlets:

$$b_{\mathbf{a},\mathbf{o}'}(\mathbf{s}', \theta) = \sum_j c_{j,\mathbf{s}'} \prod_{S', \mathbf{pa}_{S'}} \mathcal{D}_{j,\mathbf{s}'}(\theta_{S'|\mathbf{pa}_{S'}}) \tag{5}$$

◀

Note however that the number of components in the Dirichlet mixtures grows by a factor equal to the size of the state space at each time step (i.e., exponential growth with time). Hence, we consider several approximation in Section 4.2 to keep the number of components bounded.

Bayesian learning of CPDs is closed under DDNs since inference over $\Theta$ essentially learns the CPD parameters of the DDN in Figure 1. Since the CPD parameters encode transition, observation and reward distributions, the entire model is learnt simultaneously by repeatedly computing the posterior after each action-observation pair.

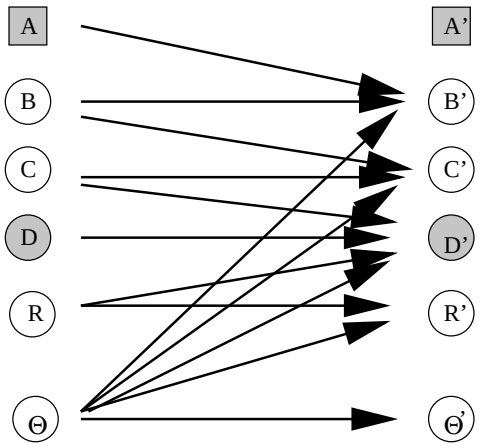


Figure 2: Dynamic decision network augmented with model variables $\mathbf{\Theta}$.

[2] Learning dynamic models is left as future work.

### 3.2 Value Function Parameterization

We now explain how optimal policies and value functions can be derived. Recall that Bayesian PORL can be modeled as a dynamic decision network (DDN) with hidden model variables $\Theta$. Conditioning each state variable on the model variables has the effect of completely determining the conditional probability distribution of each variable in the network (i.e., $\Pr(S|\mathbf{PA}_S, \Theta) = \Theta_{S|\mathbf{PA}_S}$). This important fact means that a POMDP with unknown dynamics can always be transformed in a larger POMDP with known dynamics where the unknown parameters of the original POMDP simply become hidden variables of the larger POMDP. This equivalence is quite significant since it implies that existing POMDP planning techniques can be used to optimize the policy of Bayesian PORL problems. However, the added model variables are continuous, which means that the augmented POMDP is hybrid (discrete state variables with continuous model variables). Since most POMDP planning techniques cater to discrete POMDPs, Jaulmes, Pineau et al. (2005) considered discretizing the model variables. However, the resulting number of discrete models grows exponentially with the number of model variables. Alternatively, various sampling approaches have been proposed to optimize the policy of general continuous POMDPs by forward search (Ng & Jordan 2000; Kearns, Mansour, & Ng 2002). In particular, several sampling approaches have been tailored to Bayesian reinforcement problems in fully observable domains (Dearden, Friedman, & Andre 1999; Strens 2000; Wang *et al.* 2005; Castro & Precup 2007). However, these approaches have yet to be generalized to partially observable domains and since the sample complexity grows exponentially with the search horizon, they can only perform a myopic search in practice.

In another line of research, Porta, Vlassis et al. (2006) adapted point-based value iteration techniques to general continuous POMDPs. Furthermore, the authors devised a point-based algorithm called BEETLE for Bayesian reinforcement learning in fully observable domains (Poupart *et al.* 2006). The BEETLE algorithm takes advantage of the fact that the optimal value function has a simple parameterization corresponding to the upper surface of a set of $\alpha$-functions each corresponding to a multivariate polynomial. In the rest of this section, we show that the value function for partially observable domains is also parameterized by a set of multivariate polynomials and furthermore, that these polynomials are linear combinations of Dirichlets. In the next section, we extend the BEETLE algorithm to partially observable domains.

Recall that the optimal value function $V^*$ of a POMDP satisfies Bellman's equation (Eq. 3) and is a function of $b$. In the case of Bayesian PORL, beliefs are mixtures of Dirichlets, hence it is natural to think of $V$ as a function of the mixture coefficients and the hyperparameters of the Dirichlets. Unfortunately, this parameterization generally leads to a non-convex non-linear value function, which is difficult to work with. Alternatively, we can also think of $V$ as a function of the probability of every state-model pair. While this may not seem like a good idea since there are infinitely many possible state-model pairs, the value function becomes piecewise linear and convex.

Smallwood and Sondik (1973) first showed the piecewise linear and convex properties of the optimal value function for discrete POMDPs. More precisely, the optimal value function can always be represented by the upper surface of a (possibly infinite) set $\Gamma$ of linear segments called $\alpha$-vectors (i.e., $V^*(b) = \max \alpha(b)$). The linear property stems from the fact that each $\alpha$ is a linear combination of the probability values assigned by $b$ to each state (i.e., $\alpha(b) = \sum_{\mathbf{s}} c_{\mathbf{s}} b(\mathbf{s})$). For discrete state spaces, there are finitely many states, which allows $\alpha$ to be represented as a vector of coefficients $c_{\mathbf{s}}$ for each state $\mathbf{s}$ (i.e., $\alpha(\mathbf{s}) = c_{\mathbf{s}}$), hence the name $\alpha$-vector.

The piecewise linear and convex properties of optimal value functions was also shown for Bayesian reinforcement learning in fully observable domains (Duff 2002) and general continuous POMDPs (Porta *et al.* 2006). Similar to the discrete case, the optimal value function is the upper envelope of a (possibly infinite) set $\Gamma$ of linear functionals called $\alpha$-functions. The linear property stems from the fact that $\alpha$ is an infinitesimal linear combination of the density values assigned by $b$ to each $s$ (i.e., $\alpha(b) = \int_{\mathbf{s}} c_{\mathbf{s}} b(\mathbf{s}) d\mathbf{s}$). Hence $\alpha$ can be represented as a function that returns the coefficient $c_{\mathbf{s}}$ for each state $\mathbf{s}$ of the continuous state space. The authors further showed that $\alpha$-functions are multivariate polynomials in $\Theta$ for Bayesian reinforcement learning problems in fully observable domains (Poupart *et al.* 2006). Before showing that this is also the case in partially observable domains, let's review the construction of $\alpha$-functions.

Suppose that the optimal value function $V^k(b)$ for $k$ steps-to-go is composed of a set $\Gamma^k$ of $\alpha$-functions such that $V^k(b) = \max_{\alpha \in \Gamma^k} \alpha(b)$. Using Bellman's equation, we can compute the set $\Gamma^{k+1}$ representing the optimal value function $V^{k+1}$ with $k+1$ stages-to-go. First, we rewrite Bellman's equation (Eq. 3) by substituting $V^k$ for the maximum over the $\alpha$-functions in $\Gamma^k$:

$$V^{k+1}(b) = \max_a E_b[\mathbf{R}] + \gamma \sum_{\mathbf{o}'} \Pr(\mathbf{o}'|b, \mathbf{a}) \max_{\alpha \in \Gamma^k} \alpha(b_{\mathbf{a},\mathbf{o}'})$$

Then we decompose Bellman's equation in 3 steps. The first step (Eq. 6) finds the maximal $\alpha$-function for each $\mathbf{a}$ and $\mathbf{o}'$. The second step (Eq. 7) finds the best action. The third step (Eq. 8) performs the actual Bellman backup using the maximal action and $\alpha$-functions.

$$\alpha_{b,\mathbf{a},\mathbf{o}'} = \operatorname*{argmax}_{\alpha \in \Gamma^k} \alpha(b_{\mathbf{a},\mathbf{o}'}) \tag{6}$$

$$\mathbf{a}_b = \operatorname*{argmax}_a \sum_{\mathbf{o}'} \Pr(\mathbf{o}'|b, a) \alpha_{b,\mathbf{a},\mathbf{o}'}(b_{\mathbf{a},\mathbf{o}'}) \tag{7}$$

$$V^{k+1}(b) = E_b[\mathbf{R}] + \gamma \sum_{\mathbf{o}'} \Pr(\mathbf{o}'|b, \mathbf{a}_b) \alpha_{b,\mathbf{a}_b,\mathbf{o}'}(b_{\mathbf{a}_b,\mathbf{o}'}) \tag{8}$$

We can further rewrite the third step (Eq. 8) by using $\alpha$-functions in terms of $\mathbf{s}$ and $\theta$ (instead of $b$) and expanding belief $b_{\mathbf{a}_b,\mathbf{o}'}$:

$$\begin{aligned}V^{k+1}(b) &= E_b[\mathbf{R}] + \gamma \sum_{\mathbf{o}'} \Pr(\mathbf{o}'|b,\mathbf{a}_b) \\ &\quad \sum_{\mathbf{s}'} \int_\theta b_{\mathbf{a}_b,\mathbf{o}'}(\mathbf{s}',\theta)\alpha_{b,\mathbf{a}_b,\mathbf{o}'}(\mathbf{s}',\theta)d\theta \\ &= \sum_{\mathbf{s}} \int_\theta b(\mathbf{s},\theta)\big[\sum_R [\mathbf{s}]_R \\ &\quad + \gamma \sum_{\mathbf{s}'} \Pr(\mathbf{s}'|\mathbf{s},\theta,\mathbf{a}_b)\alpha_{b,\mathbf{a}_b,[\mathbf{s}']_{\mathbf{O}'}}(\mathbf{s}',\theta)\big]d\theta\end{aligned}$$

Since the expression in the square brackets is a function of $\mathbf{s}$ and $\theta$, let's use it as the definition of an $\alpha$-function in $\Gamma^{k+1}$:

$$\alpha_b(\mathbf{s},\theta) = \sum_R [\mathbf{s}]_R + \gamma \sum_{\mathbf{s}'} \Pr(\mathbf{s}'|\mathbf{s},\theta,\mathbf{a}_b)\alpha_{b,\mathbf{a}_b,[\mathbf{s}']_{\mathbf{O}'}}(\mathbf{s}',\theta) \tag{9}$$

Hence for every $b$ we can define such an $\alpha$-function and together they form the set $\Gamma^{k+1}$. Since each $\alpha_b$ was defined by using the optimal action and $\alpha$-functions in $\Gamma^k$, then each $\alpha_b$ is necessarily optimal at $b$ and we can introduce a max over all $\alpha$-functions without changing anything:

$$V^{k+1}(b) = \sum_{\mathbf{s}} \int_\theta b(\mathbf{s},\theta)\alpha_b(\mathbf{s},\theta)d\theta \tag{10}$$

$$= \alpha_b(b) \tag{11}$$

$$= \max_{\alpha \in \Gamma^{k+1}} \alpha(b) \tag{12}$$

We are now ready to demonstrate that $\alpha$ functions are linear combinations of products of Dirichlets, which are a special class of multivariate polynomials.

**Theorem 2** *$\alpha$-functions are multivariate polynomials corresponding to linear combinations of products of Dirichlets in partially observable Bayesian RL.*

**Proof:** We give a proof by induction. Initially, $\Gamma^0$ consists of a single $\alpha$-function that assigns 0 to all beliefs. This $\alpha$-function is a trivial linear combination of Dirichlets. Assuming that $\alpha$-functions in $\Gamma^k$ are linear combinations of products of Dirichlets, we show that $\alpha_b$ in Eq. 9 is also a linear combination of products of Dirichlets.

Dropping the subscript $b$ in Eq. 9 for simplicity, we can factor $\Pr(\mathbf{s}'|\mathbf{s},\theta,\mathbf{a})$ into a product of conditional distributions, each corresponding to a model parameter (i.e., $\Pr(\mathbf{s}'|\mathbf{s},\theta,\mathbf{a}) = \prod_{S'} \theta_{[\mathbf{s}'|\mathbf{s},\mathbf{a}]_{S'|\mathbf{PA}_{S'}}}$). Furthermore, let $\alpha_{b,\mathbf{a},[\mathbf{s}']_{\mathbf{O}'}}(\mathbf{s}',\theta) = \sum_i c_{i,\mathbf{s}'} \prod_{S',\mathbf{pa}_{S'}} \mathcal{D}_{i,\mathbf{s}'}(\theta_{S'|\mathbf{pa}_{S'}})$ be a linear combination of products of Dirichlets, then Eq. 9 reads:

$$\alpha(\mathbf{s},\theta) = \sum_R [\mathbf{s}]_R + \gamma \sum_{\mathbf{s}'} [\prod_{S'} \theta_{[\mathbf{s}'|\mathbf{s},\mathbf{a}]_{S'|\mathbf{PA}_{S'}}}] \sum_i c_{i,\mathbf{s}'} \prod_{S',\mathbf{pa}_{S'}} \mathcal{D}_{i,\mathbf{s}'}(\theta_{S'|\mathbf{pa}_{S'}})$$

Since the rewards $[\mathbf{s}]_R$ are constant, they can be rewritten as scaled uniform Dirichlets. The uniform Dirichlet $\mathcal{D}(\theta;\mathbf{e})$ is a constant function in $\theta$ with all hyperparameters set to 1 (denoted by $\mathbf{e}$). Hence, we can replace $[\mathbf{s}]_R$ by $c_R D(\theta;\mathbf{e})$ where $c_R$ is a scaling factor. Furthermore, each $\theta_{[\mathbf{s}'|\mathbf{s},\mathbf{a}]_{S'|\mathbf{PA}_{S'}}}$ can be absorbed by some Dirichlet by incrementing one of its hyperparameters (i.e., $\theta_{[\mathbf{s}'|\mathbf{s},\mathbf{a}]_{S'|\mathbf{PA}_{S'}}} \mathcal{D}(\theta_{S'|\mathbf{pa}_{S'}}) = \mathcal{D}(\theta_{S'|\mathbf{pa}_{S'}};\mathbf{n}_{S'|\mathbf{pa}_{S'}} + \delta(S'|\mathbf{pa}_{S'} = [\mathbf{s}'|\mathbf{s},\mathbf{a}]_{S'|\mathbf{PA}_{S'}}))$). Performing the substitutions clearly shows that $\alpha$ is a linear combination of products of Dirichlets:

$$\alpha(\mathbf{s},\theta) = \sum_R c_R D(\theta;\mathbf{e}) + \gamma \sum_{\mathbf{s}',i} c'_{i,\mathbf{s}'} \tag{13}$$

$$\prod_{S',\mathbf{pa}_{S'}} \mathcal{D}(\theta_{S'|\mathbf{pa}_{S'}};\mathbf{n}_{S'|\mathbf{pa}_{S'}} + \delta(S'|\mathbf{pa}_{S'} = [\mathbf{s}'|\mathbf{s},\mathbf{a}]_{S'|\mathbf{PA}_{S'}})) \tag{14}$$

◀

Note however that the number of products of Dirichlets in the linear combination increases by a factor equal to the size of the state space with each Bellman backup. Hence the size of the linear combinations of the $\alpha$-functions increases exponentially with the planning horizon. We explain several approaches to circumvent this exponential blow up in Section 4.1.

# 4 Partially Observable BEETLE

The following sections explain how to extend the BEETLE algorithm to partially observable domains and how to execute the resulting policy.

## 4.1 Policy Optimization

BEETLE is a point-based value iteration algorithm adapted to Bayesian reinforcement learning in fully observable domains. It finds an approximately optimal value function represented by a set of $\alpha$-functions, each represented by a linear combination of Dirichlets. Since the optimal value function in partially observable domains has the same form, we propose a straightforward extend BEETLE, called PO-BEETLE, for partially observable domains. The pseudocode of PO-BEETLE is given in Algorithm 1. It performs point-based Bellman backups by constructing the best $\alpha_b$ at each belief $b \in B$ by following Equations 6, 7 and 9. The construction of the belief set $B$ and the termination condition are left unspecified to let practictioners make their own choice given the numerous schemes in the point-based value iteration literature. Since the size of each $\alpha$-function (i.e., number of Dirichlet components in the linear combination) may grow by a factor as large as the size of the state space, a different representation must be used or the number of components must be reduced to prevent an exponential blow up.

Although the number of components in $\alpha$-functions grows exponentially with the number of Bellman backups, $\alpha$-functions admit a *functional composition* that

**Algorithm 1** PO-BEETLE

Let $B$ be a set of reachable beliefs
$\Gamma^0 \leftarrow \{0\}$ and $i \leftarrow 0$
**repeat**
  $i \leftarrow i + 1$
  $\Gamma^i \leftarrow \emptyset$
  **for all** $b \in B$ **do**
    1. Find best $\alpha$ in $\Gamma^{i-1}$ for each $\mathbf{a}, \mathbf{o}$ (Eq. 6)
    2. Find best action $\mathbf{a}_b$ according to Eq. 7
    3. Compute $\alpha_b$ according to Eq. 9
    4. Optional: $\alpha_b \leftarrow reduce(\alpha_b)$
    5. $\Gamma^i \leftarrow \Gamma^i \cup \{(\alpha_b, \mathbf{a}_b)\}$
  **end for**
**until** some termination condition

has a linear representation with respect to the number of Bellman backups. Since $\alpha$-functions are computed recursively with Eq. 9, we can simply store how to obtain $\alpha_b$ from each $\alpha_{b,\mathbf{a}_b,\mathbf{o}'}$ at the previous iteration. In other words, $\alpha_b$ is obtained by applying some functional composition of the $\alpha$-functions at the previous iteration. So, we just need to store the functional composition. Do do this, consider the policy graph induced by the construction of the $\alpha$-functions at each Bellman backup. More precisely, think of each $\alpha$-function as a node with its associated action, which is the action found in Step 2 of Algorithm 1. Draw an edge for each observation to the node of the best $\alpha$-function found at Step 1 of Algorithm 1. Hence, at the node of $\alpha$, we can store the reward term $E_b[\mathbf{R}]$ used to compute $\alpha$ and for each edge from $\alpha$ to $\alpha_{\mathbf{a},\mathbf{o}}$ we can store the probability terms $\theta_{[\mathbf{s}'|\mathbf{s},\mathbf{a}]_{S'|\mathbf{PA}_{S'}}}$ that multiply $\alpha_{\mathbf{a},\mathbf{o}}$. Given such a policy graph, we have all the information necessary to reconstruct the linear combination corresponding to the $\alpha$-function of a node.

Hence, instead of storing exponentially large linear combinations for each $\alpha$-function, we can simply store a policy graph that is linear in the planning horizon. However, this representation may not be as convenient for operations on $\alpha$-functions. However, there is only one operation performed on $\alpha$-functions: the evaluation of a belief (i.e., $\alpha(b)$). In the case of degenerate beliefs with probability 1 for some $(\mathbf{s}, \theta)$-pair, the evaluation can be done efficiently in linear time with respect to the size of the policy graph. The idea is to perform dynamic programming by computing the value of each $\alpha$-function at the given $\theta$ for each possible state, starting from the end of the policy graph. In contrast, the evaluation of proper beliefs is not as simple since it is done by integrating over $\theta$: $\alpha(b) = \sum_s \int_\theta \alpha(\mathbf{s}, \theta) d\theta$. However we can always approximate the integral by sampling a set of $(\mathbf{s}, \theta)$-pairs and efficiently computing $\alpha(\mathbf{s}, \theta)$ with the above dynamic programming scheme.

Alternatively, we can reduce the number of Dirichlet components in a linear combination with a direct approximation. One possibility is the projection technique that we proposed in (Poupart *et al.* 2006), which is repeated in Algorithm 2 for convenience. The idea is to find the best linear combination of a fixed set of basis functions $\phi_i(\theta)$ that approximates the original linear combination by minimizing some $L_n$ norm:

$$\min_{\{c_i\}} \int_\theta |\alpha(\theta) - \sum_i c_i \phi_i(\theta)|^n d\theta$$

If we use the Euclidean norm, the coefficients $c_i$ can be found analytically by solving a linear system as described in Algorithm 2.

**Algorithm 2** Reduce(least-square-projection)

1. Let $\mu(\theta)$ be a linear combination of Dirichlets
2. Let $\phi_i(\theta)$ be basis functions $(1 \leq i \leq k)$
3. Compute $A_{i,j} = \int_\theta \phi_i(\theta)\phi_j(\theta)d\theta$ for all $i, j$
4. Compute $d_i = \int_\theta \phi_i(\theta)\mu(\theta)d\theta$ for all $i$
5. Solve $Ac = d$
6. Return $\tilde{\mu}(\theta) = \sum_i c_i \phi_i(\theta)$

Another possibility is to retain only the most important Dirichlet components as measured by the absolute value of their coefficients. This is analog to performing a beam search. Algorithm 3 describes the approach.

**Algorithm 3** Reduce(deterministic-beam)

1. Let $\mu(\theta) = \sum_i c_i \mathcal{D}_i(\theta)$ be a linear combination of Dirichlets
2. Let $c_{i_1}$ to $c_{i_k}$ be the $k$ largest coefficients in absolute value
3. Return $\tilde{\mu}(\theta) = \sum_{j=1}^k c_{i_j} \mathcal{D}_{i_j}(\theta)$

### 4.2 Policy Execution

Once a policy is optimized offline by computing a set $\Gamma$ of $(\alpha, a)$-pairs with PO-BEETLE, its execution consists of repeatedly updating the belief and executing the action associated with the best $\alpha$-function in $\Gamma$. Algorithm 4 describes the pseudocode of policy execution. Note that since the policy has already been optimized by PO-BEETLE, action selection is relatively light-weight. Nevertheless, updating the belief can become intractable since beliefs are mixtures of Dirichlets that grow exponentially with time. Similar to $\alpha$-functions, one must consider a different representation or some approximation to avoid the exponential blow up.

We can represent beliefs with a functional composition that grows linearly with time similar to the one for $\alpha$-functions. However, unlike value iteration which usually includes less than 100 steps, belief monitoring may be performed for millions of steps in some applications. Hence beliefs that grow linearly with time are impractical. As for the projection technique of Algorithm 2, it doesn't guarantee that the resulting approximate beliefs are positive in every $(\mathbf{s}, \theta)$-pair.

**Algorithm 4** Policy execution

Let $b$ be the initial belief
Let $\Gamma$ be a set of $(\alpha, \mathbf{a})$-pairs
**loop**
1. Find best $\alpha$ in $\Gamma$ for $b$
2. Execute action $\mathbf{a}$ associated with best $\alpha$
3. Receive observation $\mathbf{o}'$
4. Update belief: compute $b_{\mathbf{a},\mathbf{o}'}$ (Eq. 4)
5. Optional: $b \leftarrow reduce(b)$

**end loop**

The beam search technique of Algorithm 3 can be applied to belief updating. In fact, it is expected to perform quite well since there is only one underlying model, which can be represented by a single Dirichlet with infinitely large hyperparameters. As learning progresses, the belief will converge to the underlying model. Hence, with time, a decreasing number of Dirichlets components are expected to dominate. So dropping the least significant Dirichlet components should not hurt in the long run. However, if the Dirichlets components near the underlying model have small coefficients initially, they will systematically get pruned and the learning process may slow down or get stuck. To ensure asymptotic convergence, we propose a stochastic beam search which samples Dirichlets components according to their mixture coefficients. Algorithm 5 describes the approach.

**Algorithm 5** Reduce(stochastic-beam)

1. Let $\mu(\theta) = \sum_i c_i \mathcal{D}_i(\theta)$ be a mixture of Dirichlets
2. Sample $k$ Dirichlets $\mathcal{D}_j$ from the mixture
3. Return $\tilde{\mu}(\theta) = \sum_{j=1}^{k} D_j(\theta)$

Stochastic beam search is reminiscent of particle filtering. Hence we also consider particle filtering. The idea is to think of each Dirichlet component as a particle with a corresponding mixture probability. So, instead of generating all the Dirichlet components of a belief and pruning them by sampling with a stochastic beam, we directly sample the components to be generated. Hence particle filtering is faster than stochastic beam, yet equivalent. Algorithm 6 describes the pseudocode of particle filtering with Dirichlets particles. Note that particle filtering with $(\mathbf{s}, \theta)$-particles is not possible since there are no transition dynamics for $\theta$.

**Algorithm 6** Particle filtering

Let $b(\mathbf{s}, \theta) = \sum_i c_{i,\mathbf{s}} \mathcal{D}_{i,\mathbf{s}}(\theta)$ be the current belief
Let $\mathbf{a}$ be the action selected
Let $\mathbf{o}$ the observation received
$j \leftarrow 0$
**while** $j < k$ **do**
1. Sample $(i, \mathbf{s})$-pair according to mixture coefficients $c_{i,\mathbf{s}}$
2. Sample $\theta$ from $\mathcal{D}_{i,\mathbf{s}}(\theta)$
3. Sample $\mathbf{s}'$ from $\Pr(\mathbf{s}'|\mathbf{s}, \theta, \mathbf{a})$

**if** $\mathbf{s}'$ is consistent with $\mathbf{o}'$ **then**
4. $j \leftarrow j + 1$
5. $\mathcal{D}_{j,\mathbf{s}'}(\theta) \leftarrow \mathcal{D}_{i,\mathbf{s}}(\theta; \mathbf{n} + 1_{\mathbf{s},\mathbf{a},\mathbf{s}'})$

**end if**
**end while**
Return $\tilde{b}_{\mathbf{a},\mathbf{o}'}(\mathbf{s}', \theta) = 1/k \sum_{j=1}^{k} \mathcal{D}_{j,\mathbf{s}'}(\theta)$

## 5 Active Learning and the Exploration/Exploitation Tradeoff

Active learning occurs when the learner can influence the selection of the data from which it learns. In reinforcement learning, it is natural for the learner to be active since the action selected influences the states that will be reached in the future and therefore, the information that will be gained. An important problem of active learning is how to select actions effectively to maximize some objective, which is the sum of expected total rewards in reinforcement learning. This leads to the exploration/exploitation tradeoff where by the learner is tempted to select actions that exploit its knowledge to maximize immediate return and to select actions that explore to gain additional information to maximize future returns.

The POMDP formulation of Bayesian reinforcement learning provides a natural framework to reason about the exploration/exploitation tradeoff. Since beliefs encode all the information gained by the learner and an optimal POMDP policy is a mapping from beliefs to actions that maximizes expected total rewards (which include the immediate rewards seeked by exploitation and the future rewards seeked by exploration), it follows that an optimal POMDP policy naturally optimizes the exploration/exploitation tradeoff.

Jaulmes, Pineau et al. (2005) recently considered active learning in partially observable domains, but information gathering actions were restricted to oracles that reveal exactly the underlying state. This restriction was due to the use of a model learning approach that requires full observability of the state space. In contrast, we describe a general and principled approach to optimize active learning that does not restrict in any way the type of actions. As demonstrated in Theorem 1, belief updates with partially observable actions simply lead to mixtures of products of Dirichlets.

## 6 Conclusion and Future Work

In summary, this paper showed that beliefs and value functions can be analytically represented by a mixtures of Dirichlets and linear combinations of Dirichlets respectively. This represents an important contribution to the theory of Bayesian reinforcement learning in partially observable domains. General and principled techniques were proposed to update beliefs and optimize policies by exploiting their respective analytical form.

We are currently implementing and testing the pro-

posed algorithms and hope to report empirical results in the near future. This work could be further extended by considering the problem of structure learning (i.e., learning the graphical structure of the dynamic decision network), considering continuous states, actions and observations and considering non-stationary models.

## References


Aberdeen, D., and Baxter, J. 2002. Scaling internal-state policy-gradient methods for POMDPs. In *ICML*, 3–10.

Castro, P. S., and Precup, D. 2007. Using linear programming for Bayesian exploration in Markov decision processes. In *IJCAI*, 2437–2442.

Dearden, R.; Friedman, N.; and Andre, D. 1999. Model based Bayesian exploration. In *UAI*, 150–159.

Duff, M. 2002. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. Ph.D. Dissertation, University of Massassachusetts Amherst.

Heckerman, D. 1999. A tutorial on learning with bayesian networks. In Jordan, M., ed., *Learning in Graphical Models*. Cambridge, MA: MIT Press.

Hohl, N., and Stone, P. 2004. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *ICRA*.

Jaulmes, R.; Pineau, J.; and Precup, D. 2005. Active learning in partially observable Markov decision processes. In *ECML*, 601–608.

Kearns, M.; Mansour, Y.; and Ng, A. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning* 49:193–208.

Meuleau, N.; Peshkin, L.; Kim, K.-E.; and Kaelbling, L. P. 1999. Learning finite-state controllers for partially observable environments. In *UAI*, 427–436.

Ng, A. Y., and Jordan, M. I. 2000. PEGASUS: a policy search method for large MDPs and POMDPs. In *UAI*, 406–415.

Ng, A.; Kim, H. J.; Jordan, M.; and Sastry, S. 2003. Autonomous helicopter flight via reinforcement learning. In *NIPS*.

Ng, A.; Parr, R.; and Koller, D. 2000. Policy search via density estimation. In *NIPS*, 1022–1028.

Porta, J. M.; Vlassis, N. A.; Spaaan, M. T. J.; and Poupart, P. 2006. Point-based value iteration for continuous pomdps. *Journal of Machine Learning Research* 7:2329–2367.

Poupart, P.; Vlassis, N.; Hoey, J.; and Regan, K. 2006. An analytic solution to discrete Bayesian reinforcement learning. In *ICML*, 697–704.

Smallwood, R. D., and Sondik, E. J. 1973. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research* 21:1071–1088.

Strens, M. 2000. A Bayesian framework for reinforcement learning. In *ICML*, 943–950.

Tesauro, G. J. 1995. Temporal difference learning and TD-Gammon. *Communications of the ACM* 38:58–68.

Wang, T.; Lizotte, D.; Bowling, M.; and Schuurmans, D. 2005. Bayesian sparse sampling for on-line reward optimization. In *ICML*, 956–963.