An Analytic Solution to Discrete Bayesian Reinforcement Learning

Pascal Poupart PPOUPART@CS.UWATERLOO.CA
Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada

Nikos Vlassis VLASSIS@SCIENCE.UVA.NL
Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands

Jesse Hoey JHOEY@CS.TORONTO.EDU
Dept. of Computer Science, University of Toronto, Toronto, Ontario, Canada

Kevin Regan KMREGAN@CS.UWATERLOO.CA
Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada

Abstract

Reinforcement learning (RL) was originally proposed as a framework to allow agents to learn in an online fashion as they interact with their environment. Existing RL algorithms come short of achieving this goal because the amount of exploration required is often too costly and/or too time consuming for online learning. As a result, RL is mostly used for offline learning in simulated environments. We propose a new algorithm, called BEETLE, for effective online learning that is computationally efficient while minimizing the amount of exploration. We take a Bayesian model-based approach, framing RL as a partially observable Markov decision process. Our two main contributions are the analytical derivation that the optimal value function is the upper envelope of a set of multivariate polynomials, and an efficient point-based value iteration algorithm that exploits this simple parameterization.

1. Introduction

Over the years, reinforcement learning (RL) (Sutton & Barto, 1998) has emerged as a dominant framework for simultaneous planning and learning under uncertainty. Many problems of sequential decision making with unknown action effects can be solved by reinforcement learning (e.g., elevator scheduling (Crites & Barto, 1996), helicopter control (Ng et al., 2003), backgammon playing (Tesauro, 1995)).

Interestingly, even though RL can, in theory, enable an agent to plan and learn *online*, in practice, RL is mostly used in simulation to learn *offline*. Model-free algorithms, which directly learn an optimal policy (or value function), tend to have slow convergence, requiring too many trials to be used for online learning. In application domains where each state transition has a cost or some state transitions may lead to severe losses (e.g., helicopter crash, mobile robot collision), online learning with model-free RL is not realistic. In contrast, model-based approaches can incorporate prior knowledge to mitigate severe losses, speed up convergence and reduce the number of trials. Model-based approaches, especially Bayesian ones, can also optimally tradeoff exploration and exploitation. However model-based approaches tend to be much more complicated and computationally intensive, making them impractical for online learning.

In this paper, we derive an analytic solution to Bayesian model-based RL. While it is well known that Bayesian RL can be cast as a partially observable Markov decision process (POMDP) (Duff, 2002), the lack of a convenient parameterization for the optimal value function is one of the causes of the poor scalability of Bayesian RL algorithms. We show that for discrete Bayesian RL, the optimal value function is parameterized by a set of multivariate polynomials. This parameterization allows us to derive an efficient offline approximate policy optimization technique. Even though this optimization is done offline, learning is really done online as originally intended in the RL frame-

Appearing in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006.

work. Furthermore, online learning is not computationally intensive since it requires only belief monitoring. This removes the main concern that practitioners traditionally have with model-based approaches.

The paper is organized as follows. Sect. 2 reviews the POMDP formulation of Bayesian RL and how to do belief monitoring. Sect. 3 demonstrates that the optimal value function is the upper envelope of a set of multivariate polynomials. Sect. 4 presents an efficient algorithm called Beetle that exploits this parameterization. Sect. 5 demonstrates empirically the Beetle algorithm on a toy problem and a realistic assistive technology task. Finally, Sect. 6 concludes.

2. POMDP formulation of Bayesian RL

A Markov decision process (MDP) can be formally defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the set of states s, $\mathcal{A}$ is the set of actions a, $T(s,a,s') = \Pr(s'|s,a)$ encodes the probability that state s' is reached when action a is executed in state s, and $R(s,a,s')$ encodes the reward earned when state s' is reached after executing action a in state s. A policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is a mapping from states to actions.

The problem of reinforcement learning (RL) consists of finding an optimal policy for an MDP with a partially or completely unknown transition function. In this paper, we analytically derive a simple parameterization of the optimal value function for the Bayesian model-based approach. Bayesian learning proceeds as follows. Pick a prior distribution encoding the learner's initial belief about the possible values of each unknown parameter. Then, whenever a sampled realization of the unknown parameter is observed, update the belief to reflect the observed data. In the context of reinforcement learning, each unknown transition probability $T(s',a,s)$ is an unknown parameter $\theta^a_{s,s'}$. Since these are probabilities, the parameters $\theta_a^{s,s'}$ take values in the $[0,1]$-interval.

We can then formulate Bayesian model-based RL as a partially observable Markov decision process (POMDP) (Duff, 2002), which is formally described by a tuple $\langle \mathcal{S}_P, \mathcal{A}_P, \mathcal{O}_P, T_P, Z_P, R_P \rangle$. Here $\mathcal{S}_P = \mathcal{S} \times \{\theta_a^{s,s'}\}$ is the set of states composed of the cross product of the MDP states s with the unknown parameters $\theta_a^{s,s'}$. Since the MDP states are discrete and the unknown parameters are continuous, the POMDP state space $\mathcal{S}_P$ is hybrid. The action space $\mathcal{A}_P = \mathcal{A}$ is the same as the underlying MDP. The observation space $\mathcal{O}_P = \mathcal{S}$ consists of the MDP state space since it is fully observable. The transition function $T_P(s,\theta,a,s',\theta') = \Pr(s',\theta'|s,\theta,a)$ can be factored in two conditional distributions for the MDP states (i.e., $\Pr(s'|s,\theta_a^{s,s'},a) = \theta_a^{s,s'}$) and the unknown parameters (i.e., $\Pr(\theta'|\theta) = \delta_\theta(\theta')$ where $\delta_\theta(\theta')$ is a Kronecker delta with value 1 when $\theta' = \theta$ and value 0 otherwise). This Kronecker delta essentially denotes the assumption that unknown parameters are stationary (i.e., θ does not change). The observation function $Z_P(s',\theta',a,o) = \Pr(o|s',\theta',a)$ indicates the probability of making an observation o when state s',θ' is reached after executing action a. Since the observations are the MDP states, then $\Pr(o|s',\theta',a) = \delta_{s'}(o)$. The reward function $R_P(s,\theta,a,s',\theta') = R(s,a,s')$ is the same as the underlying MDP reward function since it doesn't depend on the unknown parameters θ,θ'.

Based on this POMDP formulation, we can learn the transition model θ by belief monitoring. At each time step, the belief (or probability density) $b(\theta) = \Pr(\theta)$ over all unknown parameters $\theta_a^{s,s'}$ is updated based on the observed transition s,a,s' using Bayes' theorem:

$$b_a^{s,s'}(\theta) = kb(\theta)\Pr(s'|\theta,s,a) \quad (1)$$

$$= kb(\theta)\theta_a^{s,s'} \quad (2)$$

In practice, belief monitoring can be performed easily when the prior and the posterior belong to the same family of distributions. If the prior b is a product of Dirichlets then the posterior $b_a^{s,s'}$ is also a product of Dirichlets since Dirichlets are conjugate priors of multinomials (DeGroot, 1970). A Dirichlet distribution $\mathcal{D}(p;n) = k\Pi_i p_i^{n_i-1}$ over a multinomial p is parameterized by positive numbers n_i, such that $n_i - 1$ can be interpreted as the number of times that the p_i-probability event has been observed. Since the unknown transition model θ is made up of one unknown distribution θ_a^s per s,a pair, let the prior be $b(\theta) = \Pi_{s,a}\mathcal{D}(\theta_a^s; n_a^s)$ such that n_a^s is a vector of hyperparameters $n_a^{s,s'}$. The posterior obtained after transition $\hat{s},\hat{a},\hat{s}'$ is:

$$b_a^{s,s'}(\theta) = k\theta_a^{s,s'}\Pi_{s,a}\mathcal{D}(\theta_a^s; n_a^s) \quad (3)$$

$$= \Pi_{s,a}\mathcal{D}(\theta_a^s; n_a^s + \delta_{\hat{s},\hat{a},\hat{s}'}(s,a,s')) \quad (4)$$

Here $\delta_{\hat{s},\hat{a},\hat{s}'}(s,a,s')$ is a Kronecker delta that returns 1 when $s = \hat{s}$, $a = \hat{a}$, $s' = \hat{s}'$, and 0 otherwise. In practice, belief monitoring is as simple as incrementing the hyperparameter corresponding to the observed transition.

3. Policy Optimization

We now explain how optimal policies and value functions can be derived. Sect. 3.1 reviews Bellman's equation for Bayesian RL. Sect. 3.2 explains how optimal POMDP solutions naturally optimize the exploration/exploitation tradeoff. Sect. 3.3 shows that the optimal value function in Bayesian RL is parameterized by a set of multivariate polynomials. This is a

key result that will be the basis of the Beetle algorithm proposed in Sect. 4.

3.1. Bellman's Equation

In POMDPs, policies π are mappings from belief states to actions (i.e., $\pi(b) = a$). The value V^π of a policy π is measured by the expected discounted sum of the rewards earned while executing it: $V^\pi(b) = \sum_{t=0}^{\infty} \gamma^t R(b_t, \pi(b_t), b_{t+1})$. An optimal policy π^* has the highest value in all belief states (i.e., $V^{\pi^*}(b) \geq V^\pi(b)\ \forall \pi, b$) and its value function $V^*(b)$ satisfies Bellman's equation:

$$V^*(b) = \max_a \sum_o \Pr(o|b,a)[R(b,a,b_a^o) + \gamma V^*(b_a^o)]. \quad (5)$$

Smallwood and Sondik (1973) showed that the optimal value function of POMDPs with discrete states is piecewise linear and convex. More precisely, it corresponds to the upper envelope of a (possibly infinite) set of linear segments $\alpha(b)$ called α-vectors.

Recall that for Bayesian RL, states are partly discrete and partly continuous, however following Duff (2002), Bellman's equation can be re-written

$$V_s^*(b) = \max_a \sum_o \Pr(o|s,b,a)[R(s,b,a,s',b_a^o) + \gamma V_{s'}^*(b_a^o)]. \quad (6)$$

Using the fact that rewards do not depend on b nor b_a^o and that observations correspond to the physical states s' in Bayesian RL, Bellman's equation can be simplified to

$$V_s^*(b) = \max_a \sum_{s'} \Pr(s'|s,b,a)[R(s,a,s') + \gamma V_{s'}^*(b_a^{s,s'})]. \quad (7)$$

Here b is the current belief in θ and $b_a^{s,s'}$ is the revised belief state according to Eq. 4.

3.2. Exploration/Exploitation Tradeoff

The POMDP formulation of Bayesian RL allows one to naturally optimize the exploration/exploitation tradeoff. Bellman's equation helps to understand why this is the case. Pure exploitation selects the action that maximizes total rewards based on b only, disregarding the fact that valuable information may be gained by observing the outcome of the action chosen. More precisely, policy optimization by pure exploitation would select actions according to the following equation, which differs from Eq. 7 only by the use of b instead of $b_a^{s,s'}$ in the right hand side:

$$V_s^*(b) = \max_a \sum_{s'} \Pr(s'|s,b,a)[R(s,a,s') + \gamma V_{s'}^*(b)] \quad (8)$$

At run time, the agent would continually perform belief monitoring to update the belief state, but the action chosen at each step would simply take into account the current belief state since the outcome of future actions hasn't been observed yet. While this may seem reasonable, it is suboptimal. Even though the outcome of future actions cannot be observed yet, we can hypothesize future action outcomes and take them into account by *conditional planning*. This is precisely what Bellman's equation (Eq. 7) achieves since all possible updated belief states $b_a^{s,s'}$ are considered with probabilities corresponding to the likelihood of reaching s'. Hence, Bellman's equation optimizes the sum of the rewards that can be derived based on the information available in b (e.g., exploitation) as well as the information gained in the future by observing the outcome of the actions selected (e.g., exploration). Alternatively, we can also argue that an optimal policy of the POMDP formulation of Bayesian RL optimizes the exploration/exploitation tradeoff simply based on the fact that such a policy maximizes the expected total return.

3.3. Value Function Parameterization

Recall that the optimal value function of POMDPs with discrete states is piecewise linear and convex (Smallwood & Sondik, 1973). More precisely, it corresponds to the upper envelope of a (possibly infinite) set Γ of linear segments $\alpha(b)$ called α-vectors (i.e., $V^*(b) = \max_{\alpha \in \Gamma} \alpha(b)$). In Bayesian RL, despite the hybrid nature of the state space, the piecewise linear and convex property still holds as demonstrated by Duff (2002) and Porta et al. (2005). The optimal value function corresponds to the upper envelope of a set Γ of linear segments called α-functions due to the continuous nature of θ (i.e., $V_s^*(b) = \max_{\alpha \in \Gamma} \alpha_s(b)$). Here α can be defined as a linear function of b subscripted by s (i.e., $\alpha_s(b)$) or as a function of θ subscripted by s (i.e., $\alpha_s(\theta)$) such that $\alpha_s(b) = \int_\theta b(\theta)\alpha_s(\theta)d\theta$. Hence, value functions in Bayesian RL can also be represented by a set of α-functions, however it is unknown how to parameterize α-functions in such a way that this parameterization be closed under Bellman backups. Due to the lack of a convenient parameterization, practitioners have had difficulty developing efficient and accurate algorithms. To date, several approximate algorithms based on confidence intervals (Kaelbling, 1993; Meuleau & Bourgine, 1999), Normal-Gamma distributions (Dearden et al., 1998), linear combinations of hyperparameters (Duff, 2003) and sampling (Dearden et al., 1999; Strens, 2000; Wang et al., 2005) have been proposed, but they tend to be computationally intensive at run time, preventing online learning or to make drastic approximations such as *myopically* optimizing the policy.

In a recent paper, Porta et al. (2005) derived that the parameterization of α-functions for continuous POMDPs with Gaussian dynamics is a linear combina-

tion of Gaussian functions. Similarly, we analytically derive that α-functions in Bayesian RL are multivariate polynomials (Theorem 1). Based on this parameterization, we propose an efficient nonmyopic approximate algorithm called Beetle in Sect. 4.

Before establishing our main theorem, let us first review the Bellman backup operator and the updating of the α-functions (Duff, 2002). Suppose that the optimal value function $V_s^k(b)$ for k steps-to-go is composed of a set Γ^k of α-functions such that $V_s^k(b) = \max_{\alpha\in\Gamma^k} \alpha_s(b)$. Using Bellman's equation, we can compute by dynamic programming the best set Γ^{k+1} representing the optimal value function V^{k+1} with $k+1$ stages-to-go. First we rewrite Bellman's equation (Eq. 7) by substituting V^k for the maximum over the α-functions in Γ^k:

$$V_s^{k+1}(b) = \max_a \sum_{s'} \Pr(s'|s,b,a)[R(s,a,s') + \gamma \max_{\alpha\in\Gamma^k} \alpha_{s'}(b_a^{s,s'})].$$

Then we decompose Bellman's equation in 3 steps. The first step (Eq. 9) finds the maximal α-function for each a and s'. The second step (Eq. 10) finds the best action a. The third step (Eq. 11) performs the actual Bellman backup using the maximal action and α-functions.

$$\alpha_{b,a}^{s,s'} = \operatorname*{argmax}_{\alpha\in\Gamma^k} \alpha_{s'}(b_a^{s,s'}) \quad (9)$$

$$a_b^s = \operatorname*{argmax}_a \sum_{s'} \Pr(s'|s,b,a)[R(s,a,s') + \gamma\alpha_{b,a}^{s,s'}(b_a^{s,s'})] \quad (10)$$

$$V_s^{k+1}(b) = \sum_{s'} \Pr(s'|s,b,a_b^s)[R(s,a_b^s,s') + \gamma\alpha_{b,a_b^s}^{s,s'}(b_{a_b^s}^{s,s'})] \quad (11)$$

We can further rewrite the third step (Eq. 11) by using α-functions in terms of θ (instead of b) and expanding the belief state $b_{a_b^s}^{s,s'}$:

$$\sum_{s'} \Pr(s'|s,b,a_b^s)[R(s,a_b^s,s') + \gamma \int_\theta b_{a_b^s}^{s,s'}(\theta)\alpha_{b,a_b^s}^{s,s'}(\theta)d\theta] \quad (12)$$

$$= \sum_{s'} \int_\theta b(\theta)\Pr(s'|s,\theta,a_b^s)[R(s,a_b^s,s') + \gamma\alpha_{b,a_b^s}^{s,s'}(\theta)d\theta] \quad (13)$$

$$= \int_\theta b(\theta)\Big[\sum_{s'} \Pr(s'|s,\theta,a_b^s)[R(s,a_b^s,s') + \gamma\alpha_{b,a_b^s}^{s,s'}(\theta)]\Big]d\theta \quad (14)$$

Since the expression in the outer square brackets is a function of s and θ, let's use it as the definition of an α-function in Γ^{k+1}:

$$\alpha_{b,s}(\theta) = \sum_{s'} \Pr(s'|s,\theta,a_b^s)[R(s,a_b^s,s') + \gamma\alpha_{b,a_b^s}^{s,s'}(\theta)]. \quad (15)$$

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

$$V_s^{k+1}(b) = \int_\theta b(\theta)\alpha_{b,s}(\theta)d\theta \quad (16)$$

$$= \alpha_{b,s}(b) \quad (17)$$

$$= \max_{\alpha\in\Gamma^{k+1}} \alpha_s(b) \quad (18)$$

We are now ready to prove our main theorem:

Theorem 1 *α-functions in Bayesian RL are multivariate polynomials.*

Proof: We give a proof by induction. Initially, Γ^0 consists of a single α-function that assigns 0 to all belief states. This α-function is a trivial multivariate polynomial. Assuming α-functions in Γ^k are multivariate polynomials, we show that $\alpha_{b,s}$ in Eq. 15 is also a multivariate polynomial.

In Eq. 15, we can substitute $\Pr(s'|s,\theta,a)$ by $\theta_a^{s,s'}$, where for simplicity we use a to denote the optimal action a_b^s. Let $\alpha_{b,a}^{s,s'}(\theta) = \sum_i c_{i,s'}\mu_{i,s'}(\theta)$ where $\mu_{i,s'}(\theta) = \Pi_{\hat{s},\hat{a},\hat{s}'}(\theta_{\hat{a}}^{\hat{s},\hat{s}'})^{\lambda_{\hat{a},i}^{\hat{s},\hat{s}'}}$ is a monomial over the parameter space, with non-negative powers λ (one for each parameter). The indices s, a, s' have a "hat" to distinguish them from those used in the definition of $\alpha_{b,s}$. The reward $R(s,a,s')$ can also be written as a degenerate constant monomial $c_{s'}\mu_{s'}(\theta)$ such that $c_{s'} = R(s,a,s')$ and $\mu_{s'}(\theta) = \Pi_{\hat{s},\hat{a},\hat{s}'}(\theta_{\hat{a}}^{\hat{s},\hat{s}'})^0$. Eq. 15 then reads:

$$\alpha_{b,s}(\theta) = \sum_{s'} \theta_a^{s,s'}[c_{s'}\mu_{s'}(\theta) + \gamma\sum_i c_{i,s'}\mu_{i,s'}(\theta)] \quad (19)$$

We can absorb $\theta_a^{s,s'}$ into the monomials by incrementing the appropriate powers. If we write

$$\mu'(\theta) = \Pi_{\hat{s},\hat{a},\hat{s}'}(\theta_{\hat{a}}^{\hat{s},\hat{s}'})^{(\lambda_{\hat{a}}^{\hat{s},\hat{s}'} + \delta_{s,a,s'}(\hat{s},\hat{a},\hat{s}'))} \quad (20)$$

then Equation 15 reads:

$$\alpha_{b,s}(\theta) = \sum_{s'}[c_{s'}\mu'_{s'}(\theta) + \gamma\sum_i c_{i,s'}\mu'_{i,s'}(\theta)] \quad (21)$$

which is again a multivariate polynomial. ◀

4. The Beetle Algorithm

Since multivariate polynomials form a closed representation for α-functions under Bellman backups, we propose a simple and efficient point-based value iteration algorithm for Bayesian RL called Beetle (i.e., Bayesian Exploration Exploitation Tradeoff in LEarning).

4.1. Point-based value iteration

The Beetle algorithm is an extension of the Perseus algorithm (Spaan & Vlassis, 2005) for Bayesian RL. First, a set of reachable s,b pairs is sampled by simulating several runs of a default or random policy. Then (approximate) value iteration is done by performing point-based backups at those sampled s,b pairs based on Eq. 9, 10 and 15. For a given s,b pair, the best α-function for each a,s' is computed according to Eq. 9.

Then, the optimal action is computed according to Eq. 10. A new α-function is constructed according to Eq. 15. This new α-function is represented very simply by the non-negative powers λ of its monomial terms.

As is, Beetle suffers from an important source of intractability. At each backup, the number of terms of the multivariate polynomial of the resulting α-function grows significantly. More precisely, in Eq. 21, the number of monomials is multiplied by $O(|\mathcal{S}|)$, which yields a number of monomials that grows exponentially with the planning horizon.

4.2. α-function Projection

In order to mitigate the exponential growth in the number of monomials, after each Bellman backup we project each new α-function onto a multivariate polynomial with a smaller number of monomials. Intuitively, finding a good projection can be cast as an optimization problem where we would like to simultaneously minimize the error at each θ. For instance, when projecting an α-function onto a linear combination of monomial basis functions (i.e., $\sum_i c_i \phi_i(\theta)$), minimizing an L_n norm yields:

$$\min_{\{c_i\}} \int_\theta |\alpha(\theta) - \sum_i c_i \phi_i(\theta)|^n d\theta \tag{22}$$

If we use a Euclidean norm, the optimal coefficients c_i can be found analytically by solving a system of linear equations $Ax = d$ where $A_{i,j} = \int_\theta \phi_i(\theta)\phi_j(\theta)d\theta$, $d_i = \int_\theta \phi_i(\theta)\alpha(\theta)d\theta$ and $x_j = c_j$.

Alternatively, since α-functions can be defined with respect to θ or b, we can also devise a projection scheme that minimizes error at some belief points instead of all θ's. When using an L_n norm, this can be done by minimizing $\int_b |\alpha(b) - \sum_i c_i \phi_i(b)|^n db$. Since the integral over b is generally difficult to compute and in many domains only a small region of belief space is visited, minimizing the error only at a sample B of reachable belief points is more practical:

$$\min_{\{c_i\}} \sum_{b \in B} |\alpha(b) - \sum_i c_i \phi_i(b)|^n \tag{23}$$

If we use a Euclidean norm, the optimal coefficients c_i can be found analytically by solving a system of linear equations $Ax = d$ where $A_{i,j} = \sum_{b \in B} \phi_i(b)\phi_j(b)$, $d_i = \sum_{b \in B} \phi_i(b)\alpha(b)$ and $x_j = c_j$.

In practice, the optimization of Eq. 22 with a Euclidean norm is faster computationally and approximates uniformly at all θ's. In contrast, if a good set B of reachable belief states is used, then the optimization of Eq. 23 may yield a better approximation since it focuses on the reachable belief states. Note that the optimization of Eq. 23 indirectly minimizes error at all θ's in a weighted fashion since belief points are densities over θ, whereas the optimization of Eq. 22 minimizes error at all θ's uniformly.

Ideally, we would like to pick basis functions as close as possible to the monomials of α-functions. When comparing the equations for belief monitoring (Eq. 4) and backing up α-functions (Eq. 11) it is interesting to note that in both cases, powers are incremented with each s, a, s' transition. Hence belief states and α-functions are both made up of similar monomials. Hence we propose to use the set of reachable belief states generated at the beginning of the Beetle algorithm as the set of basis functions.

Note that a fixed basis set also allows us to precompute several operations to reduce computation during point-based backups. More precisely, for each point-based backup, the α-functions of the previous step are all defined with respect to the same components (but different coefficients). Then the actual backup always transforms those components in the same way by incrementing some hyperparameters. Hence we can pre-compute the projection of each backed-up component.

So we can represent α-functions in a very compact way just by a column vector $\tilde{\alpha}$ corresponding to the coefficients of the fixed basis functions. We can also precompute a projected transition function $\tilde{T}_a^{s,s'}$ in matrix form for each s, a, s'. Similarly we can pre-compute the projection of the reward function and store basis coefficients in column vectors $\tilde{R}_a^{s,s'}$ for each s, a, s'. Altogether, point-based backups can be performed by simple matrix operations. For instance, Eq. 15 becomes

$$\tilde{\alpha}_{b,s} = \sum_{s'} \tilde{T}_a^{s,s'} [\tilde{R}_a^{s,s'} + \gamma \tilde{\alpha}_{b,a}^{s,s'}]. \tag{24}$$

4.3. Parameter Tying

In classic reinforcement learning, the entire transition dynamics are unknown. With the above Bayesian RL formulation, this leads to an unknown distribution θ_a^s for every state-action pair. When the number of states and actions are large, the amount of computation and the amount of interaction with the environment both become prohibitive.

Fortunately, in practice, the transition dynamics are rarely completely unknown. Sometimes, just a few transition probabilities are unknown. In other situations, several unknown transition probabilities are known to be the same (allowing parameter tying). More generally, the transition dynamics may be jointly expressed as a function of a small number of parameters (e.g. factored models). In addition to being able to encode the uncertainty with a small number of unknowns, the amount of interaction for online learning may be significantly reduced by starting with *informative* priors, that is prior distributions skewed towards a small range of values (i.e., low entropy).

Note that the Beetle algorithm can be used directly when unknown parameters are tied. We simply have one θ_i per *different* unknown distribution. When the transition dynamics are factored using a dynamic Bayesian network representation, our Beetle algorithm can again be used directly. In this case, the unknowns are the conditional probability distributions. Hence, we have one θ_i per unknown conditional distribution. Note that the probability of transitioning to s' from s when executing a is now the product of several conditional probabilities. Hence, for each observed transition s, a, s', we increment several powers, one per conditional probability table, during belief monitoring as well as point-based backups. In all cases, α-functions remain multivariate polynomials.

4.4. Reward Function

So far we have assumed that the reward function is known. We argue that this is not a restriction. The Beetle algorithm can still learn reward functions with a finite number of possible values. By considering a factored model, we can treat the reward signal r as a state variable. The reward function $R(s, a, s') = r$ can then be encoded as a conditional probability distribution $\Pr(r|s, a, s')$ which can be learned like all the other conditional probability distributions. In the case of a continuous reward signal, a sufficiently fine discretization should provide enough accuracy.

4.5. Discussion

Traditionally, Bayesian RL was considered too complex and intractable to be of practical use. This paper actually shows that the optimal value function has a simple analytical form consisting of a set of multivariate polynomials. This analytical form allows us to derive an efficient point-based value iteration algorithm. As a result, we can optimize a policy offline. This optimization can be efficient as long as the number of unknowns remains small. As argued in the previous section, the transition dynamics of many problems can be encoded with few parameters by tying parameters together or using a factored model. Note that for effective online learning, what really matters is the computation time while executing the policy, not the time for offline optimization. In many domains such as robotics, elevator control and assistive technologies, it is quite acceptable to have a computer in the lab spend a few hours to optimize the policy by running Beetle before downloading it into an agent for execution. However, at run time, actions must often be selected in a fraction of a second for realtime execution. Beetle can easily achieve this since belief monitoring and action selection are not computationally intensive.

While policy optimization is done offline, it is important to realize that learning is really done online. The policy computed consists of a mapping from state-belief pairs to actions. Even though this mapping is fixed throughout its execution, the belief states change with each state transition. Recall that belief monitoring is essentially the process by which the unknown transition dynamics are learned. Hence, the policy indirectly adapts with each belief update. The main drawback of offline policy optimization is that the precomputed policy must cater to as many scenarios as possible. In theory, it should prescribe an optimal action for every belief state, but this is usually intractable. Hence, we have to settle for a suboptimal policy that is hopefully good at the belief states that are more likely to be visited, and hopefully generalizes well over the remaining belief states via the use of α-functions. To that effect, point-based value iteration concentrates its effort on finding good actions at a sample of reachable belief states. Note that this idea is also used in classic RL approaches with value function approximation.

5. Experiments

We consider two problems. The first is the toy "chain" problem used in (Strens, 2000; Dearden et al., 1998), while the second comes from a realistic assistive technology scenario (Boger et al., 2005). In both problems, we experiment with varying degrees of parameter tying and we evaluate our methods by comparing them to two heuristic methods:

EXPLOIT This is a strictly online method with no offline optimization, which purely exploits its current belief at each step. We simply monitor the belief state online and pick the best action by solving the MDP for the *expected* model (i.e., average belief). While this method is simple, it tends to be slow at run time since an MDP must be solved between each action executed and it suffers from a lack of exploration.

DISCRETE_POMDP An alternative to Beetle is to discretize the unknown distributions θ in N values and to build a discrete POMDP, which can be solved using Perseus (Spaan & Vlassis, 2005). The drawback of this approach is the exponential explosion of the state space, which consists of the cross product of the physical states with N discrete values for each unknown distribution (i.e, $O(|\mathcal{S}|N^k)$ for k unknown distributions).

5.1. Problem Descriptions

Fig. 1(a) shows the "chain" problem from (Strens, 2000; Dearden et al., 1998), in which the agent has two

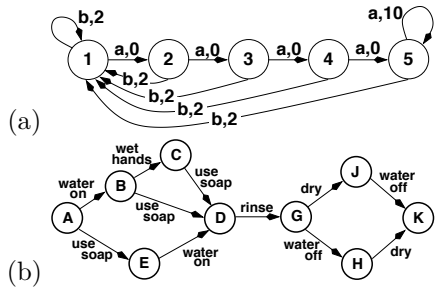

Figure 1. (a) Chain problem showing the action, reward for each transition (b) plansteps for the handwashing problem.

actions *a,b* which cause transitions between five states. At each time step, the agent "slips" and performs the opposite action with probability $p_{slip} = 0.2$.

A more realistic problem domain is concerned with assisting persons with cognitive disabilities complete activities of daily living such as handwashing. We consider a simplified version of the system developed by Boger et al. (2005) that gives audio prompts to help users wash their hands with minimal assistance from a caregiver. Since the level of independence varies widely depending on each user, a major issue is to learn user characteristics that influence their ability to carry out the task. Note that those characteristics can only be learned online as the system interacts with each user. Framed as a Bayesian RL problem, we want the system to learn user types as quickly as possible since it can be quite frustrating for users to be given inappropriate prompts. The states of the handwashing problem can be grouped into nine *plansteps* shown in Fig. 1(b). The system has two actions available: to do nothing, or to issue an audio prompt corresponding to the current *planstep*. A user of the system will exhibit certain *behaviors*: doing nothing, doing the best possible action at a *planstep*, doing the second best action (if there are two choices), or regressing (e.g. putting soap on their hands after they are clean at *planstep=g*). Each user has some distribution over these behaviors, which may depend on both the current *planstep*, and the action of the system. Typically, the system's prompt will increase the probability that the user will perform the best action for a *planstep*.

5.2. Results

Table 1 shows the results from the chain and handwashing problems. We experimented with 3 structural priors referred as `tied`, `semi` and `full`. The `full` version corresponds to the extreme (and perhaps rare) case where the dynamics are completely unknown. In navigation scenarios such as the chain problem, the effects of each action are usually known up to some noise term (i.e., the slip probability). Similarly, in assistive scenarios such as handwashing, system effects are usually known (or can be learned through simulation) except for user behaviors. Hence, more realistic encodings of the chain and handwashing problems assume that the transition dynamics are known except for the slip and behavior probabilities, which are state and action independent in the `tied` version, while action dependent in the `semi-tied` version.

We report the expected total return (averaged over 500 runs) with standard deviation for the first 1000 steps (without any discounting) for the exploit and discrete POMDP heuristics, and Beetle. In all cases, 30 Bellman iterations were performed and 2000 belief points were sampled for Beetle and the discrete POMDP heuristic. The first 200 (linearly independent) belief points were selected as basis functions for Beetle. The initial belief state is a uniform Dirichlet. For the discrete POMDP heuristic, the continuous space of each unknown distribution θ_a^s was discretized into 100 grid points selected at random uniformly.

The optimal return given the true model is reported as a (utopic) upper bound. Beetle found near optimal policies for the `tied` and `semi-tied` versions, while doing poorly on the `full` version. Since the dynamics are completely unknown in the `full` version, Beetle has trouble pre-computing a policy that is good for *all* possible models. Beetle found statistically equivalent or better policies when compared to the discrete POMDP heuristic, which found very good policies for the tied and semi-tied versions, while running out of memory for the full versions. This confirms that discretizing is impractical for problems with many free parameters. The exploit heuristic finds provably optimal policies for the `tied` version since there is no exploration required (i.e., the unknown distributions are tied across all actions). However, exploration is required for the `semi-tied` and `full` versions, which explains the suboptimal performance of the exploit heuristic. For further comparison, Dearden et al. (1998) and Strens (2000) report results for several other Bayesian RL heuristics on the `chain_full` problem, the best of which, "Bayesian DP" (similar to the exploit heuristic in that actions are selected greedily with respect to a model sampled from the current belief instead of the expected model) scored 3158 ± 31.

The running times for our Matlab implementation of Beetle are reported in the last two columns. We also wrote a C implementation (which is almost complete at the time of publication) that achieves running times one to two orders of magnitude faster. The second last

problem	$\|\mathcal{S}\|$	$\|\mathcal{A}\|$	free params	optimal (utopic)	discrete POMDP	exploit	Beetle	Beetle time (minutes) precomputation	optimization
chain_tied	5	2	1	3677	3661 ± 27	3642 ± 43	3650 ± 41	0.4	1.5
chain_semi	5	2	2	3677	3651 ± 32	3257 ± 124	3648 ± 41	1.3	1.3
chain_full	5	2	40	3677	na-m	3078 ± 49	1754 ± 42	14.8	18.0
handw_tied	9	2	4	1153	1149 ± 12	1133 ± 12	1146 ± 12	2.6	11.8
handw_semi	9	2	8	1153	990 ± 8	991 ± 31	1082 ± 17	3.4	52.3
handw_full	9	6	270	1083	na-m	297 ± 10	385 ± 10	125.3	8.3

Table 1. Expected total reward for chain and handwashing problems. na-m indicates insufficient memory.

prior	0	10	20	30
chain_f	1754 ± 42	3453 ± 47	2034 ± 57	3656 ± 32
hand_s	1082 ± 17	1056 ± 18	1097 ± 17	1106 ± 16
hand_f	385 ± 10	540 ± 10	1056 ± 12	1056 ± 12

Table 2. Expected total reward for varying priors

column indicates the time used to precompute projected transition and reward functions by minimizing error with respect to all $\theta's$ (Eq. 22). The last column reports the time to optimize by Beetle with the projected transition and reward functions. Recall that precomputation and optimization times are borne offline and therefore are in an acceptable range. Action selection takes less than 0.3 seconds.

We also tested Beetle with *informative* priors in Table 2. Instead of starting Beetle with a uniform prior (i.e, counts set to 1), we tried more informative priors by varying a parameter k from 0 to 30. That is, the Dirichlet counts are set to 1 plus k times the probabilities of the true model. As k increases, the confidence in the true model increases. In scenarios where we have some belief about the transition probabilities, but we are not completely sure, we can reduce the model uncertainty by using such an informative prior. On the problems for which Beetle didn't find a near optimal policy with a uniform prior, Table 2 shows that increasingly informative priors generally improve Beetle's performance since it can focus on finding a good policy for a smaller range of likely models.

6. Conclusion

In this paper, we have shown that optimal value functions for Bayesian RL are parameterized by sets of multivariate polynomials, and exploited this parameterization to develop an effective algorithm called Beetle. It naturally optimizes the exploration/exploitation tradeoff. It allows practitioners to easily encode prior knowledge, which permits Beetle to focus only on the truly unknown parts of the dynamics, reducing the amount of exploration necessary. Furthermore, online efficiency is achieved by precomputing offline a policy and doing only action selection and belief monitoring at run time. Overall, this work represents an important step towards the development of effective *online* RL algorithms.

We plan to extend this work on Bayesian RL in several directions, including continuous state, action and observation spaces, partially observable domains and multi-agent systems. We also plan to explore how to handle and possibly learn non-stationary dynamics.

References

Boger, J., Poupart, P., Hoey, J., Boutilier, C., Fernie, G., & Mihailidis, A. (2005). A decision-theoretic approach to task assistance for persons with dementia. *IJCAI* (pp. 1293–1299).

Crites, R. H., & Barto, A. G. (1996). Improving elevator performance using reinforcement learning. *NIPS* (pp. 1017–1023).

Dearden, R., Friedman, N., & Andre, D. (1999). Model based Bayesian exploration. *UAI* (pp. 150–159).

Dearden, R., Friedman, N., & Russell, S. (1998). Bayesian Q-learning. *AAAI* (pp. 761–768).

DeGroot, M. H. (1970). *Optimal statistical decisions*. New York: McGraw-Hill.

Duff, M. (2002). *Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes*. Doctoral dissertation, University of Massassachusetts Amherst.

Duff, M. (2003). Design for an optimal probe. *ICML* (pp. 131–138).

Kaelbling, L. P. (1993). *Learning in embedded systems*. MIT Press.

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

Ng, A., Kim, H. J., Jordan, M., & Sastry, S. (2003). Autonomous helicopter flight via reinforcement learning. *NIPS*.

Porta, J. M., Spaan, M. T., & Vlassis, N. (2005). Robot planning in partially observable continuous domains. *Proc. Robotics: Science and Systems*.

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

Spaan, M. T. J., & Vlassis, N. (2005). Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, *24*, 195–220.

Strens, M. (2000). A Bayesian framework for reinforcement learning. *ICML*.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning*. Cambridge, MA: MIT Press.

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

Wang, T., Lizotte, D., Bowling, M., & Schuurmans, D. (2005). Bayesian sparse sampling for on-line reward optimization. *ICML*.