Appl Intell (2013) 39:345–353
DOI 10.1007/s10489-012-0416-2

Monte-Carlo tree search for Bayesian reinforcement learning

Ngo Anh Vien · Wolfgang Ertel · Viet-Hung Dang · TaeChoong Chung

Published online: 22 February 2013

Abstract Bayesian model-based reinforcement learning can be formulated as a partially observable Markov decision process (POMDP) to provide a principled framework for optimally balancing exploitation and exploration. Then, a POMDP solver can be used to solve the problem. If the prior distribution over the environment's dynamics is a product of Dirichlet distributions, the POMDP's optimal value function can be represented using a set of multivariate polynomials. Unfortunately, the size of the polynomials grows exponentially with the problem horizon. In this paper, we examine the use of an online Monte-Carlo tree search (MCTS) algorithm for large POMDPs, to solve the Bayesian reinforcement learning problem online. We will show that such an algorithm successfully searches for a near-optimal policy. In addition, we examine the use of a parameter tying method to keep the model search space small, and propose the use of nested mixture of tied models to increase robustness of the method when our prior information does not allow us to specify the structure of tied models exactly. Experiments show that the proposed methods substantially improve scalability of current Bayesian reinforcement learning methods.

Keywords Bayesian reinforcement learning · Model-based reinforcement learning · Monte-Carlo tree search · POMDP

N.A. Vien (✉) · W. Ertel
Institute of Artificial Intelligence,
Ravensburg-Weingarten University of Applied Sciences,
Weingarten 88250, Germany
e-mail: ngoa@hs-weingarten.de

W. Ertel
e-mail: ertel@hs-weingarten.de

V.-H. Dang
Research and Development Center for Science and Technology,
DuyTan University, Da Nang, Vietnam
e-mail: dangviethungha@gmail.com

T.C. Chung
Department of Computer Engineering, Kyung Hee University,
Seoul, South Korea
e-mail: tcchung@khu.ac.kr

1 Introduction

Reinforcement learning (RL) [33] provides a framework for simultaneously acting and learning in unknown environments. To act well in such situations, a reinforcement learning algorithm has to handle the exploration-exploitation trade-off—it needs to balance actions that reduce its uncertainty about the environment with actions that exploit what it already knows. RL has had some remarkable practical successes in various areas, including learning to play checkers [28], backgammon [35–37], job-scheduling [42], chess [4], dynamic channel allocation [30, 38], and others [14, 15, 17, 22, 23].

Traditionally, RL algorithms can be divided into two major approaches: model-free and model-based. Model-free approaches attempt to directly learn the optimal policy by approximating the cost-to-go of each state, called a value function. These methods often have large variance and poor trade-off between exploration/exploitation. On the other hand, model-based approaches attempt to learn a model of the environment, then compute the optimal policy based on that learnt model. These approaches normally have better trade-off between exploration/exploitation. However both of them are impractical to learn online due to intensive computation and poor trade-off ability. One approach to mitigate this problem is to use Bayesian model-based RL [6, 8, 24–26, 41]. Because it will trade-off exploration/exploitation, and uses less data required.

Bayesian reinforcement learning can be represented as a partially observable Markov decision process (POMDP) problem [8]. Its policy is a mapping from the posterior distribution (or history of observations) to an action. This POMDP problem can be solved by an online planning algorithm. In addition, its policy at each step can be considered as a suggestion of the appropriate action for online Bayesian reinforcement learning. When representing Bayesian reinforcement learning as a POMDP, the posterior distribution of parameters given an observation is often conveniently represented in closed form as a product of Dirichlet distributions. Under this condition, it was shown in [24] that the optimal value function in Bayesian reinforcement learning can be represented using a set of multivariate polynomials. Unfortunately, the size of the polynomial set grows exponentially with the problem horizon, severely limiting the applicability of the method.

In order to exploit this closed representation, the BEETLE algorithm was proposed in [24] which is a belief-lookahead approach allowing to derive an offline policy by doing approximate policy optimization, then learns the model online using the offline policy. More precisely, the offline optimization task is to solve the planning problem of the POMDP formulation. Nevertheless, this method still lacks scalability of Bayesian RL algorithms because it suffers from a source of intractability. The intractability source is mostly from the exponential growth of the number of monomials in α-functions' representation. In [24], the authors also proposed one approximation method to mitigate this critical intractability. This method tries to initialize one set of monomial basis function, then projects α-functions onto a linear combination of the elements in that set. The fixed basis set only accelerated the policy computation, but it yielded a poor sub-optimal policy.

In this paper, we examine an application of the *partially Observable Monte-Carlo Planning* (POMCP) method [29] as an online solver for Bayesian reinforcement learning. POMCP uses Monte-Carlo sampling to overcome the curse of dimensionality during look-ahead planning, and it uses particle filters to approximate beliefs in very large or continuous state spaces. However, we exploit properties of the Dirichlet distribution to maintain the belief for Bayesian reinforcement learning in closed form and propose the use of nested mixtures of such distributions to provide robustness. For action selection, POMCP uses the bandit algorithm UCB1 [3] for selective sampling of actions during rollout. On the other hand, POMCP solves the exploration-exploitation dilemma by doing Monte-Carlo tree search (MCTS) from the current belief node. The Bayesian RL's POMDP has a continuous and highly-dimensional state space which makes its belief space have a large covering number, a measure of the size of the belief space [16]. To achieve a small covering number, parameter tying, where the transition functions for different states are assumed to be the same, is often used. This requires strong assumption on which states have the same parameter values. To reduce the need for such strong assumptions, we propose the use of nested mixture of tied models so that good results can be obtained if our assumptions are correct, but robust performance can still be obtained if our assumptions are not perfect. The nested mixtures also have closed form belief representation, if the tied models are products of Dirichlets. Experiments show that the algorithm scales better than the existing state of the art method and that the nested mixture model is able to provide robust performance when the model structures are not known exactly. The experimental results report both ways of applying POMCP: using particle filter and Dirichlet distribution to maintain the belief. The results of nested mixture models is only for the way using Dirichlet distribution.

In the following, we describe some related works. Then, we start with the POMDP formulation of Bayesian RL in Sect. 3. Next, we describe the POMCP algorithm, and its application for the Bayesian RL in Sect. 4 and Sect. 5, respectively. The nested mixtures of tied models are presented in Sect. 6. Then, experimental results are presented in Sect. 7. Finally, we conclude with some discussions on the future research in Sect. 8.

2 Related work

Bayesian reinforcement learning has been studied intensively in many works such as [2, 7–10, 12, 13, 24, 32, 39, 41]. Of these, the BEETLE [24] and BFS3 [2] algorithms are the closest to ours. BEETLE uses an offline point-based POMDP solver to find a policy that can then be used online as a reinforcement learning algorithm. BEETLE's representation of the value function grows exponentially with the planning horizon, while we provide an online approach which does not need to explicitly represent a value function and policy. Furthermore, our algorithm scales up efficiently with problems having larger transition matrices.

Bayesian Sparse Sampling [41] is a modification of the Sparse Sampling method [19] that applies only in the Bayesian setting. At each belief node, an action is chosen by solving an MDP model sampled from the posterior. This action-selection strategy done by the exact-solve step would slow down the algorithm and limit its applicability for large domains. Another approach (BFS3) [2], also inspired from Sparse Sampling method [19], uses Forward Search Sparse Sampling [40], which is one particular Monte-Carlo tree search (MCTS) algorithm. It also preferentially expands the search tree by maintaining hard upper and lower bounds on the values for each state and action so as to direct the rollouts: action is chosen greedily according to the upper bound, and the next state is chosen based on the uncertain.

There is an alternative approach in RL, called PAC-MDP, efficient in dealing with the trade-off between exploration/exploitation [5, 18, 19, 31, 34]. PAC-MDP algorithms use exploration actions gathering necessary information, then later exploit this information to choose optimal or near-optimal actions, which maximize the cumulative reward. Recently, Bayesian methods combined with PAC-MDP approach were also developed to build a better exploration model [1, 21]. These methods were proved to give lower sample complexity bounds.

The algorithm that we use in this paper is a modification of the partially Observable Monte-Carlo Planning (POMCP) method in [29]. POMCP uses particle filtering to approximate beliefs for solving continuous or very large POMDPs in [29]. In this paper, we exploit properties of the Dirichlet distribution to maintain the belief for Bayesian reinforcement learning in closed form and propose the use of nested mixtures of such distributions to provide robustness.

3 POMDP formulation of Bayesian RL

3.1 POMDP formulation

In this paper, we assume to only consider a RL problem whose environment is formulated as a discrete Markov decision process (MDP). A discrete MDP is defined by $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S}$ is a discrete state space, $\mathcal{A}$ is a discrete action space, $\mathcal{T}(s, a, s') = P(s'|s, a)$ defines the probability of a next state, $\mathcal{R}(s, a, s')$ defines an immediate reward. RL algorithms try to find an optimal policy $\pi : \mathcal{S} \to \mathcal{A}$ with an assumption of unknown environment dynamics in order to maximize the expected total discounted reward $\mathrm{E}(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s'_t))$, where s_t, a_t, s'_t, and γ denote a state, an action, a next state, and a discount factor respectively.

Bayesian RL learning algorithm can be formulated into a POMDP $\langle \mathcal{S}_P, \mathcal{A}, \mathcal{T}_P, \mathcal{R}_P, \mathcal{O}, \mathcal{Z} \rangle$ [8, 24] in which its state space $\mathcal{S}_P$ consists of the underlying MDP's state space $\mathcal{S}$ as well as the parameter space (unknown environment dynamics), and its observation space $\mathcal{O}$ is the state space $\mathcal{S}$ of the underlying MDP. Specifically, if each unknown transition probability is parameterized by a parameter $\theta^a_{s,s'} \in [0, 1]$, then the new state space is $\mathcal{S}_P = \mathcal{S} \times \{\theta^a_{s,s'}\}$. The action space $\mathcal{A}$ is kept similarly, the observation space $\mathcal{O} = \mathcal{S}$ which is the observable MDP state space. The transition $T_P(s, \theta, a, s', \theta')$ is defined as in Eq. (1)

$$\begin{aligned} T_P\big(s, \theta, a, s', \theta'\big) &= \Pr\big(s', \theta' \big| s, \theta, a\big) \\ &= \Pr\big(s' \big| s, \theta^{s,s'}_a, a\big) \\ &= \theta^{s,s'}_a \delta_\theta\big(\theta'\big) \end{aligned} \tag{1}$$

where δ is the Kronecker delta. The observation function $\mathcal{Z}(s', a, o) = \Pr(o|s', a)$ is defined as $\Pr(o|s', a) = \delta_{s'}(o)$. The reward function $R_P(s, \theta, a, s', \theta') = R(s, a, s')$ as in the original MDP's reward function. Thus, we obtained a continuous state, discrete observation POMDP problem.

The POMDP state is partially observable and monitored as a belief. Let the prior belief over all unknown parameters θ^s_a be $b(\theta) = \Pr(\theta)$. Assuming that the prior belief is a product of Dirichlets, then the posterior is also a product of Dirichlets. More specifically, the belief is written as

$$b(\theta) = \prod_{s,a} \mathcal{D}\big(\theta^s_a; n^s_a\big) \tag{2}$$

where each unknown distribution θ^s_a per one pair (s, a) is represented by one Dirichlet $\mathcal{D}(\theta^s_a; n^s_a) = k \prod_{s'} \theta^{n^{s,s'}_a - 1}_{s,a,s'}$; and n^s_a is a vector of parameters $\{n^{s,s'}_a\}$. Then, the closed form of the belief update operator after observing a transition $(\bar{s}, \bar{a}, \bar{s}')$ is

$$\begin{aligned} b^{s,s'}_a(\theta) &= k \theta^{s,s'}_a \prod_{s,a} \mathcal{D}\big(\theta^s_a; n^s_a\big) \\ &= \prod_{s,a} \mathcal{D}\big(\theta^s_a; n^s_a + \delta_{\bar{s},\bar{a},\bar{s}'}\big(s, a, s'\big)\big) \end{aligned} \tag{3}$$

Thus, the effect of a belief update is just to increase the count corresponding to the observed transition.

3.2 BEETLE algorithm

In this section, we will describe briefly one analytic algorithm to the discrete Bayesian RL which is called the BEETLE algorithm and introduced in [24]. Recall that the POMDP has a state space of partly observable discrete component (current observable MDP state s) and partly unobservable continuous component (parameter θ). Then, the Bellman's update can be written in the form of

$$V^{t+1}_s(b) = \max_a \sum_o \Pr(o|b, a)\big[R(b, a) + \gamma V^t\big(b^o_a\big)\big] \tag{4}$$

where b^o_a defines the next belief at a current belief b taking action a and observing o. In POMDP formulation, o is the observation which is in the state space of the original MDP, thus we can rewrite Eq.(4) as

$$V^{t+1}_s(b) = \max_a \sum_{s'} \Pr\big(s'\big|s, b, a\big)\big[R\big(s, a, s'\big) + \gamma V^t_{s'}\big(b^{s,s'}_a\big)\big] \tag{5}$$

The optimal value function can be represented by a set Γ of α-functions, and each α-function is a multivariate polyno-

mial. For more detail, assuming that the optimal value function is a piecewise-linear convex function and written as

$$V_s^{t+1}(b) = \max_{\alpha\in\Gamma} \int_\theta \alpha_s(\theta)b(\theta)d\theta = \max_{\alpha\in\Gamma^{t+1}} \alpha_s(b) \quad (6)$$

where $\alpha \in \Gamma^{t+1}$ is a function over parameter space of θ, and has the following Bellman update from Γ^t

$$\alpha_{b,s}^{t+1}(\theta) = \sum_{s'} \Pr(s'|s,\theta,a)\big[R(s,a,s') + \gamma\alpha_{b,a}^{s,s'}(\theta)\big] \quad (7)$$

where $\alpha_{b,a}^{s,s'} = \arg\max_{\alpha\in\ \Gamma^t} \alpha_{s'}(b_a^{s,s'})$.

If Γ^0 is an initial set of α-functions consisting of an unique element, such that $\alpha(\theta) = 0$ for all θ, which is trivially a multivariate polynomial. Using induction and Eq. (7), then α^t is a multivariate polynomial, if α^{t-1} is a multivariate polynomial. This is the major result of BEETLE algorithm [24].

However, we observe that the number of monomials in this representation increase in a factor of $O(|\mathcal{S}|)$ (see Eq. (7)). Thus, this representation causes the main intractability in Bayesian RL. In the next section, we will introduce one approximation algorithm which has the capability of efficiently dealing with this problem. We will implicitly represent α-functions as a finite state controller.

4 Partially observable Monte-Carlo planning (POMCP)

The POMCP algorithm [29] is an online planning method that extends the Monte-Carlo tree search (MTCS) method [20] to POMDPs. Each node of the tree search is represented by a pair of the value of history h and the count of times that history h has been visited $T(h) = \langle V(h), N(h)\rangle$; where $V(h)$ is estimated by the mean return of Monte-Carlo simulations starting from h. The tree is considered as a search tree of visited histories, whose root is the initial belief b_0. For each possible action a, a value $Q(h,a)$ is estimated also by Monte-Carlo simulations, and associated with a visit count $N(h,a)$. Clearly, the total count must be $N(h) = \sum_a N(h,a)$. Each simulation starts from a state sampled from the belief $b(h)$, chooses actions based on the multi-armed bandit algorithm UCB1 [3]. If child nodes for all actions of the current node are added into the tree, actions are selected to maximize an upper confidence bound on the action value,

$$Q^{\oplus}(h,a) = Q(h,a) + c\sqrt{\frac{\log N(h)}{N(h,a)}}$$

$$\pi(h) = \operatorname*{argmax}_a\ Q^{\oplus}(h,a)$$

where c is a bias parameter which defines the proportion of exploitation and exploration. If $c = 0$, the UCB1 policy becomes a greedy policy.

Otherwise, if not all child nodes for all actions of the current node are added into the tree, the rollout action is used to select actions. The POMCP policy consists of two stages. Initially, it exploits the knowledge contained in the search tree. Once the simulation leaves the scope of the search tree, it must select actions according to a rollout policy. After each simulation, one new node, corresponding to the first new updated history h^{ao} (assume action a is taken at h, then observing an o), is added into the search tree, with initial value $\langle V_0(h^{ao}), N_0(h^{ao})\rangle$. As more simulations are taken, the POMCP policy improves with more added nodes, and the Monte-Carlo estimates become more accurate.

5 Monte-Carlo planning for Bayesian RL

Partially Observable Monte-Carlo Planning (POMCP) was introduced in [29] for solving continuous or large POMDPs. It uses particle filter for belief representation and a UCT search method [20] to select actions online. Instead of using a particle filter, we represent the belief in a closed form using a Dirichlet distribution. The POMCP algorithm uses a random policy for the rollout policy, because it has no knowledge beyond the scope of the search tree. Our algorithm is Monte-Carlo Bayesian Reinforcement Learning (MCBRL) as described in Algorithm 1. Each node of the search tree is labeled with a pair of a current MDP state and a history $T(s,h) = \langle V(s,h), N(s,h)\rangle$, the belief state is $\mathcal{B}(\theta,s,h)$. Each simulation starts from the MDP state s and is simulated with a MDP model θ sampled from $\mathcal{B}(\cdot,s,h)$.

For one estimation step, MCBRL algorithm will query the oracle (POMDP model of the unknown MDP) at most $N \times d$ times comparing to $t \times d \times |A|^2 \times C^2$ times of Sparse Sampling methods such as Bayesian Forward Search Sparse Sampling (BFS3) [40]; where t is the number of trajectories that will be simulated, d is the maximum depth of any simulated trajectories, $|A|$ is the number of actions available, and C is the number of times each action is tried for a given node in the tree. The parameter t and C of BFS3 are considered as the number of simulations N of MCBRL. On the other hand, the method like Bayesian Sparse Sampling [41] even has to solve exactly a sampled MDP at each action-selection step within the inner loop. This shows that the computational power required makes BFS3 and Bayesian Sparse Sampling methods of limited applicability for larger domains.

Our MCBRL algorithm also converges to the optimal value function like POMCP. We now provide the way to derive a MDP with histories as states as in [29]. Lemma 1 is a straightforward derivation from [29]. By this Lemma, the convergence analysis of MCBRL is equivalent to POCMP [29] and UCT [20].

Algorithm 1 Monte-Carlo online planning Algorithm for Bayesian RL

procedure ONLINESEARCH(s, h)

for $i = 1$ to N **do**
  Sample a model $\theta \sim \mathcal{B}(h)$.
  SIMULATE($s, \theta, h, 0$).
end for
return $\operatorname{argmax}_a Q(s, h, a)$.

procedure SIMULATE(s, θ, h, d)

if $(\gamma^d < \epsilon)$ **then**
  return 0
end if
if $(h \notin T)$ **then**
  for all $a \in \mathcal{A}$ **do**
    Add a node (s, θ, ha, d) into the tree T.
  end for
  return ROLLOUT(s, θ, h, d)
end if
Select an action $a \leftarrow \operatorname{argmax}_b \{Q(hb) + c\sqrt{\frac{\log N(h)}{N(hb)}}\}$.
Take action a, and observe $(o, r) \sim \theta$.
$R \leftarrow r + \gamma$.SIMULATE($o, \theta, hao, d + 1$)
$N(h) \leftarrow N(h) + 1$, $N(ha) \leftarrow N(ha) + 1$
$Q(ha) \leftarrow Q(ha) + \frac{R - Q(ha)}{N(ha)}$
return R

procedure ROLLOUT(s, θ, h, d)

if $(\gamma^d < \epsilon)$ **then**
  return 0
end if
Select an action $a \sim \pi_{rollout}(s, h, \cdot)$
Take action a, and observe $(o, r) \sim \theta$.
return $r + \gamma$.ROLLOUT($o, \theta, hao, d + 1$)

Lemma 1 *Given a POMDP $\langle \mathcal{S}_P, \mathcal{A}, \mathcal{T}_P, \mathcal{R}_P, \mathcal{O}, \mathcal{Z} \rangle$ which is the derived POMDP of a Bayesian RL problem. The derived MDP of this POMDP with histories as states is $\tilde{\mathcal{M}} = (\mathcal{H}, \mathcal{A}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}})$, where*

$$\tilde{\mathcal{T}}^a_{h,hao} = \int_{s \in \mathcal{S}_P} \int_{s' \in \mathcal{S}_P} \mathcal{B}(s, h) \mathcal{T}_P\big(s' | s, a\big) \mathcal{Z}\big(o | s', a\big) ds ds'$$

and,

$$\tilde{\mathcal{R}}^a_h = \int_{s \in \mathcal{S}_P} \mathcal{B}(s, h) \mathcal{R}_P(s, a) ds$$

Then the value function of the derived MDP is equivalent to the value function of the derived POMDP of a Bayesian RL problem, $\tilde{V}^\pi(h) = V^\pi(h), \forall \pi$.

6 Nested mixtures of tied models

In practice, the parameters θ^s_a are often the same for many states and can be tied together to reduce the number of parameters. Sometimes we do not have enough knowledge to know the correct states to tie together. In such cases, we may still be able to specify a nested sequence of tied models $M_1 \subseteq M_2 \subseteq \cdots \subseteq M_n$, where M_n is the model without any tied parameters. Each tying model M_i is presumably predefined as in [24]. We then use a mixture of model

$$p(\theta) = \sum_i^n \beta_i \, p(\theta | M_i) \tag{8}$$

as the prior, where $\beta_i = p(M_i)$ is the respective weight of each model M_i. If the true model is in a tied model M_i with a small i, we would like to do well. Otherwise, we would still like to have adequate performance. Assuming that θ is the parameter of a multinomial distribution and each mixture component $p(\theta | M_i)$ is a tied model drawn from the product of Dirichlets.

We need to keep track of the posterior distribution as

$$p(\theta | \text{data}) = \sum_i p(M_i | \text{data}) p(\theta | M_i, \text{data}). \tag{9}$$

The term $p(\theta | M_i, \text{data})$ can be computed as before. In order to compute the model's weight posterior distribution

$$p(M_i | \text{data}) \propto p(M_i) p(\text{data} | M_i) \tag{10}$$

we need compute $p(\text{data} | M_i)$ by marginalization over parameters as

$$\begin{aligned} p(\text{data} | M_i) &= \int_\theta p(\text{data}, \theta | M_i) d\theta \\ &= \int_\theta p(\text{data} | \theta) p(\theta | \gamma_{M_i}) d\theta \\ &= \prod_c \int_{\theta^c} \prod_{s \in c} \prod_a p\big(o^{(s,a)} | \theta^c\big) p\big(\theta^c | \gamma_{M_i}\big) d\theta^c \end{aligned}$$

where $o^{(s,a)}$ is the observed outcome counts for action a in state s, γ_{M_i} is the Dirichlet distribution parameter and c is a cluster within M_i, then

$$\begin{aligned} p(\text{data} | M_i) = &\prod_{c,a} \frac{\Gamma(\sum_j \gamma_j)}{\prod_j \Gamma(\gamma_j)} \\ &\times \frac{\prod_s \Gamma(\sum_j o^{s,a}_j + 1)}{\prod_{j,s} \Gamma(o^{s,a}_j + 1)} \frac{\prod_j \Gamma(\sum_s o^{s,a}_j + \gamma_j)}{\Gamma(\sum_{j,s} o^{s,a}_j + \gamma_j)} \end{aligned}$$

(which is a multivariate Polya distribution).

We know that the belief $B(h)$ over a history h is equivalent to the posterior distribution of the model's parameters

$p(\theta|\text{data} = h)$ as computed in Eq. (9). Thus, in order to apply this nested mixture model for the MCBRL algorithm, the model sampling step (line 2 of ONLINESEARCH procedure in Algorithm 1) changes as following: We first sample a model as $M_i \sim p(M_i|\text{data})$ as computed in Eq. (10), then sample the dynamics from $\theta^{s'}_{s,a} \sim p(\theta|M_i, \text{data})$ as before in Sect. 5.

7 Experiments

In this section, we evaluate the performance of MCBRL on two simple problems from the previous works: "Chain" [1, 7, 24] and 6 × 6 maze [1, 27], and compare with BEETLE [24], BOSS [1], and BFS3 [2] algorithms.

7.1 Chain problem

Figure 1(a) shows the *Chain* problem which has five states and two actions, $\mathcal{A} = \{a, b\}$. The actions usually causes an agent to move along the direction shown in the figure, but the agent *slips* (move in the other direction) with probability 0.2.

The experiments are implemented with 3 structural priors: Tied, Semi-tied, and Full. The Full version means that the dynamics are completely unknown. 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. For all experiments, MCBRL uses the exploration constant c set as in [29], the search horizon is computed by setting a discount factor $\gamma = 0.95$ and an accuracy $\epsilon = 0.01$. The number of simulation is set to 1000.

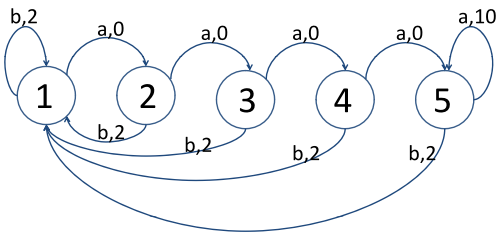

Fig. 1 Chain problem showing action, reward for each transition

Table 1 shows the results of MCBRL compared to other algorithms from the chain problem. The default policy is the *Exploit* policy which is the MDP policy for the average model from the current belief [24]. The results without discounting are averaged over 500 runs with standard deviation for the first 1000 steps. The results show that MCBRL is as effective as BFS3, BOSS and BEETLE in Tied and Semi setting. *Exploit* policy is myopic and does not implement any exploration, so its performance is worse than MCBRL's in Semi and Full settings. Moreover, MCBRL can also be effective in Full setting in which BEETLE fails to do more careful exploration due to the large parameter space (40-dimensional parameter space). However, MCBRL still does exploration less effectively than BOSS and BFS3 due to the large parameter space. BFS3 performs better because it uses *a sparse sampling* algorithm and directs the rollouts by maintaining upper and lower bounds on the values for each state and action. So it searches more efficiently, especially in large state spaces. The continuous and high-dimensional parameter space is still one of most challenging problems for any POMDP solvers. We have tried the *Exploit* policy [11] to replace the rollout policy (with informed rollout policy), but the performance decreases. The poorer performance of MCBRL with informed rollout policy is consistent with the discovery in [11].

We also test MCBRL with the use of a particle filter, which does not need a closed form representations of belief. The model belief is updated approximately using an unweighted particle filter, which is similar to the POMCP algorithm [29]. The performance of MCBRL with particle filter is near the performance of MCBRL with exact belief update (however with a constrained prior assumption) in Tied and Semi-Tied settings. However its performance in Full setting degrades, because the parameters space is very large when using a small sample size (1000 particles) for the particle filter.

In order to evaluate the mixture of parameter tyings in Sect. 6, we use a variant of the chain problem as in [1], named Chain2. Chain2 problem includes one more cluster.

Table 1 Cumulative reward for Chain problem

	Tied	Semi	Full
BEETLE	3650 ± 41	3648 ± 41	1754 ± 42
Exploit	3642 ± 43	3257 ± 124	3078 ± 49
BOSS	3657	3651	3003
BFS3	3655 ± 27	3652 ± 27	3055 ± 29
MCBRL	3653 ± 32	3650 ± 34	2675 ± 35
MCBRL (informed policy)	3579 ± 35	3571 ± 34	2498 ± 41
MCBRL (particle filter)	3613 ± 31	3520 ± 35	1579 ± 44

Cluster 1 (states 1, 3, 5) is similar to the only cluster in the original Chain. Cluster 2 (states 2,4) is with distribution 0.3/0.7 for action a, and reverse for action b. Chain2 is following to the Semi-tied setting. Moreover, we manually build 3 mixtures: Tied (has one parameter, two clusters are considered the same), Semi-tied (has two parameters in order to separately capture two clusters), and Full setting for Chain2. We also use three mixtures (Tied, Semi, Full) for the Chain problem. The value of the optimal policy for Chain2 is 3301. The result tells that the mixture model works well when the Tied model fails in Chain2 whose true model is Semi-tied. The cumulative reward of the mixture model for Chain2 is 3164. And, if we use Semi-tied model, we obtain 3241. To summarize, we put the results of two problems in Table 2. The results show that MCBRL with nested mixture models can quickly find and learn the correct tied model as described in Fig. 2. The performance of MCBRL with nested mixture models is near optimal because there is a true model in the mixture model set. However, if there is no true model in the set, MCBRL can still find the closest model in the set to the true one.

Table 2 Results for Chain & Chain2

	Chain	Chain 2
Tied	3653 ± 32	1673 ± 34
Semi	3650 ± 34	3241 ± 34
Full	2675 ± 53	2475 ± 41
Mixture	3650 ± 39	3164 ± 45

7.2 Maze problem

In this section, we experiment with the 6×6 maze problem as in [1] and [27]. In this problem, shown in Fig. 3, there are 4 possible actions {N, S, E, W}. The agent starts at the cell labeled S, and has to find the goal cell labeled G. The shaded cells are pits. Each action has 80 % intended effect and 20 % noisy effect which makes the agent move in one of the two perpendicular directions. Each step receives -0.001 reward, and terminal rewards of -1.0 and 1.0 for falling into a pit or reaching the goal, respectively.

We run MCBRL with 1000 sampling simulations, a discount factor $\gamma = 0.997$, the horizon accuracy $\epsilon = 0.0001$ (about 3065 steps). All results are averaged over 50 trials of 500 episodes. We observe that if we cluster the maze problem based on the behavior of each pair (s, a), we obtain optimally only 5 clusters among 16 possible ways (based on

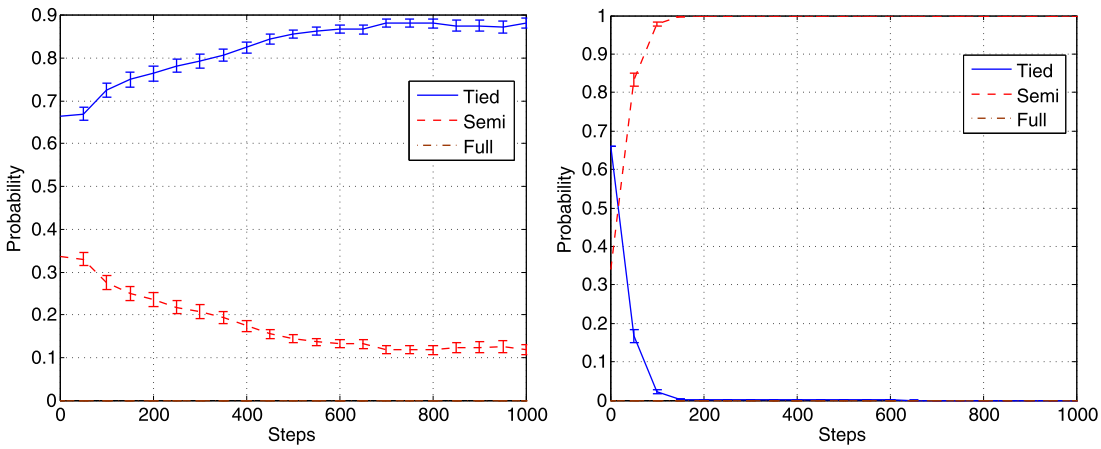

Fig. 2 The probabilities of each tied model in (*left*) Chain and (*right*) Chain2 problems. The *error bars* are standard deviations

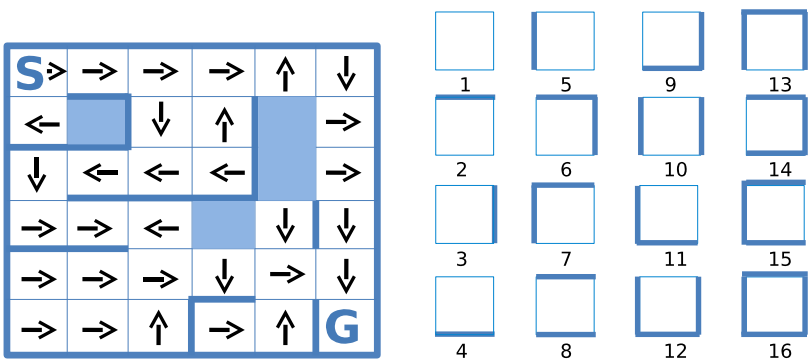

Fig. 3 6×6 maze problem with an optimal policy (*on the left*) and 16 clusters (*on the right*)

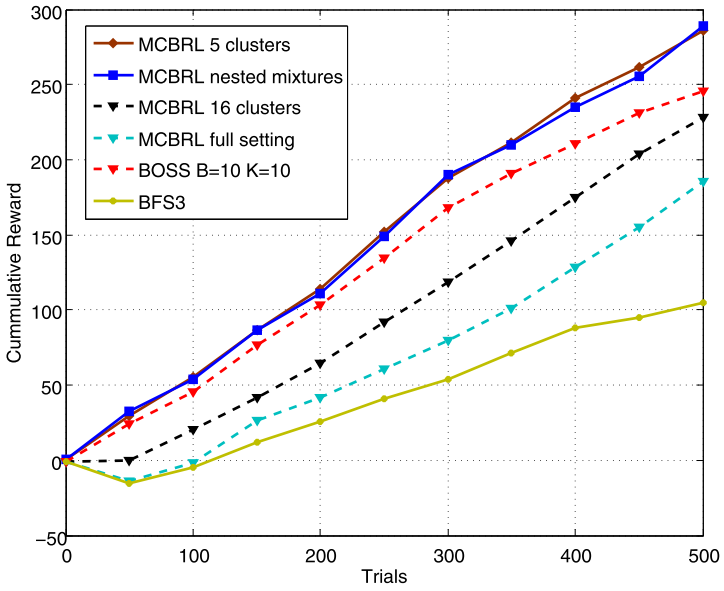

Fig. 4 Cumulative training reward vs. trials for 6×6 maze

combinations of walls around one pair) of clustering them. The right panel on Fig. 3 depicts the configuration of the clustering of 16 clusters. The 5 clustering model can be obtained if we do a partition as $\{1\}, \{2, 3, 4, 5\}, \{6, 7, 9, 11\}, \{8, 10\}, \{12, 13, 14, 15\}$ and not use 16. First, we evaluate the performance of MCBRL without nested mixture models. The MCBRL is experimented with three manually built tying models: 5 clusters, 16 clusters, and full setting (without clustering). Second, MCBRL is experimented with a nested mixture model of those tying models.

The cumulative rewards obtained by BOSS [1], BFS3 [2], MCBRL with three separate models, and MCBRL with the nested mixture model (Sect. 6) are shown in Fig. 4. Our MCBRL algorithm with the nested mixture model performs better than BOSS (with nonparametric model clustering), which is able to find only about 10 clusters, and BFS3. This is because the MCBRL algorithm uses a fixed set of tying models which makes it quickly find and learn the best one (in terms of capturing the problem's correct dynamics) with only 5 clusters. BFS3 uses *sparse sampling*, so it uses more memory than our method. Unfortunately, our computer does not have enough memory to construct such a large tree with BFS3. We used Student's t-test to verify the hypothesis that the performance of MCBRL with nested mixture models is better than BOSS and BFS3. The test's results guarantee that the performance of MCBRL is significantly better than BOSS until the 100th episode and BFS3 until the 50th episode, with a confidence level greater than 99.5 %

8 Conclusion

We examined the use of partially observable Monte-Carlo planning for online solving of Bayesian reinforcement learning problems. The use of online Monte-Carlo simulation avoids one source of intractability in offline Bayesian reinforcement learning methods—the exponential growth of the value function representation with time horizon. We further propose the use of a nested mixture of tied models as a method for increasing the robustness of the method when the structure of the parameter space is not known well. Experiments show that the method performs well and substantially increases the scalability of current solvers.

We have only studied the use of the method for learning MDPs. It would be interesting to extend the method to learning POMDPs. The lack of a compact representation of beliefs appears to be one obstacle for extending the method to POMDPs. It may be interesting to examine approximate methods such as particle filters for belief representation in these problems.

Acknowledgements This work was supported by the Collaborative Center of Applied Research on Service Robotics (ZAFH Servicerobotik, http://www.zafh-servicerobotik.de) and the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2010-0012609).

References

1. Asmuth J, Li L, Littman ML, Nouri A, Wingate D (2009) A Bayesian sampling approach to exploration in reinforcement learning. In: Proceedings of the 25th conference on uncertainty in artificial intelligence (UAI-09)
2. Asmuth J, Littman ML (2011) Learning is planning: near Bayes-optimal reinforcement learning via Monte-Carlo tree search. In: Proceedings of the twenty-seventh conference on uncertainty in artificial intelligence, pp 19–26
3. Auer P, Cesa-Bianchi N, Fischer P (2002) Finite-time analysis of the multiarmed bandit problem. Mach Learn 47(2–3):235–256

4. Baxter J, Tridgell A, Weaver L (2000) Learning to play chess using temporal differences. Mach Learn 40(3):243–263
5. Brafman RI, Tennenholtz M (2002) R-max—a general polynomial time algorithm for near-optimal reinforcement learning. J Mach Learn Res 3:213–231
6. Castro PS, Precup D (2007) Using linear programming for Bayesian exploration in Markov decision processes. In: IJCAI 2007. Proceedings of the 20th international joint conference on artificial intelligence, Hyderabad, India, January 6–12, 2007, pp 2437–2442
7. Dearden R, Friedman N, Russell SJ (1998) Bayesian Q-learning. In: Proceedings of the fifteenth national conference on artificial intelligence and tenth innovative applications of artificial intelligence conference, AAAI/IAAI 98, Madison, WI, USA, July 26–30, 1998, pp 761–768
8. Duff M (2002) Optimal learning: computational procedures for Bayes-adaptive Markov decision processes. PhD thesis, University of Massassachusetts Amherst
9. Engel Y, Mannor S, Meir R (2003) Bayes meets bellman: the Gaussian process approach to temporal difference learning. In: International conference on machine learning (ICML), pp 154–161
10. Engel Y, Mannor S, Meir R (2005) Reinforcement learning with Gaussian processes. In: International conference on machine learning (ICML), pp 201–208
11. Gelly S, Silver D (2007) Combining online and offline knowledge in uct. In: International conference on machine learning (ICML), pp 273–280
12. Ghavamzadeh M, Engel Y (2006) Bayesian policy gradient algorithms. In: Advances in neural information processing (NIPS), pp 457–464
13. Ghavamzadeh M, Engel Y (2007) Bayesian actor-critic algorithms. In: International conference on machine learning (ICML), pp 297–304
14. Granmo OC, Glimsdal S (2012) Accelerated Bayesian learning for decentralized two-armed bandit based decision making with applications to the goore game. Appl Intell
15. Hong J, Prabhu VV (2004) Distributed reinforcement learning control for batch sequencing and sizing in just-in-time manufacturing systems. Appl Intell 20(1):71–87
16. Hsu D, Lee WS, Rong N (2007) What makes some POMDP problems easy to approximate? In: Advances in neural information processing (NIPS)
17. Iglesias A, Martínez P, Aler R, Fernández F (2009) Learning teaching strategies in an adaptive and intelligent educational system through reinforcement learning. Appl Intell 31(1):89–106
18. Kakade S, Kearns MJ, Langford J (2003) Exploration in metric state spaces. In: International conference on machine learning (ICML), pp 306–312
19. Kearns MJ, Singh SP (2002) Near-optimal reinforcement learning in polynomial time. Mach Learn 49(2–3):209–232
20. Kocsis L, Szepesvári C (2006) Bandit based Monte-Carlo planning. In: European conference on machine learning (ECML), pp 282–293
21. Kolter JZ, Ng AY (2009) Near-Bayesian exploration in polynomial time. In: International conference on machine learning (ICML), p 65
22. Li J, Li Z, Chen J (2011) Microassembly path planning using reinforcement learning for improving positioning accuracy of a 1 cm^3 omni-directional mobile microrobot. Appl Intell 34(2):211–225
23. Pakizeh E, Palhang M, Pedram MM (2012) Multi-criteria expertness based cooperative Q-learning. Appl Intell
24. Poupart P, Vlassis NA, Hoey J, Regan K (2006) An analytic solution to discrete Bayesian reinforcement learning. In: International conference on machine learning (ICML), pp 697–704
25. Ross S, Chaib-draa B, Pineau J (2007) Bayes-adaptive POMDPs. In: Advances in neural information processing (NIPS)
26. Ross S, Pineau J (2008) Model-based Bayesian reinforcement learning in large structured domains. In: Proceedings of the 24th conference in uncertainty in artificial intelligence, pp 476–483
27. Russell SJ, Norvig P (2003) Artificial intelligence: a modern approach, 2nd edn. Prentice Hall, Upper Saddle River
28. Samuel AL (1959) Some studies in machine learning using the game of checkers. IBM J Res Dev 3(3):210–229
29. Silver D, Veness J (2010) Monte-Carlo planning in large POMDPs. In: Advances in neural information processing (NIPS), pp 2164–2172
30. Singh SP, Bertsekas D (1996) Reinforcement learning for dynamic channel allocation in cellular telephone systems. In: Advances in neural information processing systems, vol NIPS, pp 974–980
31. Strehl AL, Littman ML (2008) An analysis of model-based interval estimation for Markov decision processes. J Comput Syst Sci 74(8):1309–1331
32. Strens MJA (2000) A Bayesian framework for reinforcement learning. In: Proceedings of the seventeenth international conference on machine learning (ICML 2000). Stanford University, Stanford, CA, USA, June 29–July 2, 2000, pp 943–950
33. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press, Cambridge
34. Szita I, Szepesvári C (2010) Model-based reinforcement learning with nearly tight exploration complexity bounds. In: International conference on machine learning (ICML), pp 1031–1038
35. Tesauro G (1992) Practical issues in temporal difference learning. Mach Learn 8:257–277
36. Tesauro G (1994) Td-gammon, a self-teaching backgammon program, achieves master-level play. Neural Comput 6(2):215–219
37. Tesauro G (1995) Temporal difference learning and td-gammon. Commun ACM 38(3):58–68
38. Vien NA, Viet NH, Lee S, Chung T (2009) Policy gradient SMDP for resource allocation and routing in integrated services networks. IEICE Trans 92-B(6):2008–2022
39. Vien NA, Yu H, Chung T (2011) Hessian matrix distribution for Bayesian policy gradient reinforcement learning. Inf Sci 181(9):1671–1685
40. Walsh TJ, Goschin S, Littman ML (2010) Integrating sample-based planning and model-based reinforcement learning. In: Proceedings of the twenty-fourth AAAI conference on artificial intelligence (AAAI 2010), Atlanta, GA, USA, July 11–15, 2010, pp 11–15
41. Wang T, Lizotte DJ, Bowling MH, Schuurmans D (2005) Bayesian sparse sampling for on-line reward optimization. In: International conference on machine learning (ICML), pp 956–963
42. Zhang W, Dietterich TG (1995) A reinforcement learning approach to job-shop scheduling. In: International joint conferences on artificial intelligence, pp 1114–1120