2012 11th International Conference on Machine Learning and Applications

Monte Carlo Tree Search for Bayesian Reinforcement Learning

Ngo Anh Vien
Institute of Artificial Intelligence
Ravensburg-Weingarten University of Applied Sciences
Weingarten 88250, Germany
Email: ngo@hs-weingarten.de

Wolfgang Ertel
Institute of Artificial Intelligence
Ravensburg-Weingarten University of Applied Sciences
Weingarten 88250, Germany
Email: ertel@hs-weingarten.de

***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**

I. Introduction

Traditionally, reinforcement learning (RL) [1] algorithms can be divided into two major approaches: model-free and model-based. Model-free approaches 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 [2], [3], [4], [5], [6]. Because, these methods can optimally trade-off exploration/exploitation, and use less data required.

Bayesian model-based reinforcement learning can be represented as a partially observable Markov decision process (POMDP) problem [7]. 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 observations is often conveniently represented in closed form as a product of Dirichlet distributions. Under this condition, it was shown in [4] 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, BEETLE algorithm was proposed in [4] 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 of 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 this paper, we examine an application of the *partially Observable Monte-Carlo Planning* (POMCP) method [8] 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. The Bayesian RL's POMDP has a continuous and highly-dimensional state space which make its belief space have a large covering number, a measure of the size of the belief space [9]. 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

978-0-7695-4913-2/12 $26.00 © 2012 IEEE
DOI 10.1109/ICMLA.2012.30

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.

II. POMDP Formulation of Bayesian RL

A. 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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$ with an assumption of unknown environment dynamics in order to maximize the expected total discounted reward $\mathrm{E}\big(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s'_t)\big)$, 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 $\langle \mathcal{S}_P, \mathcal{A}, \mathcal{T}_P, \mathcal{R}_P, \mathcal{O}, \mathcal{Z} \rangle$ [7], [4] 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 $T_P(s,\theta,a,s',\theta') = \Pr(s',\theta'|s,\theta,a)$ which is factored in two parts $\Pr(s'|s,\theta^{s,s'}_a,a) = \theta^{s,s'}_a$ and $\Pr(\theta'|\theta) = \delta_\theta(\theta')$ (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 $R_P(s,\theta,a,s',\theta') = R(s,a,s')$ is 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

$$b(\theta) = \prod_{s,a} \mathcal{D}(\theta^s_a; n^s_a) \tag{1}$$

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

$$\begin{aligned} b^{s,s'}_a(\theta) &= k\theta^{s,s'}_a \prod_{s,a} \mathcal{D}(\theta^s_a; n^s_a) \\ &= \prod_{s,a} \mathcal{D}(\theta^s_a; n^s_a + \delta_{\bar{s},\bar{a},\bar{s}'}(s,a,s')) \end{aligned} \tag{2}$$

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

B. BEETLE Algorithm

In this section, we will describe briefly one analytic algorithm to the discrete Bayesian RL which is called BEETLE algorithm and introduced in [4]. 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

$$V^{t+1}_s(b) = \max_a \sum_o \Pr(o|b,a)[R(b,a) + \gamma V^t(b^o_a)] \tag{3}$$

where b^o_a defines the next belief if at a current belief b taking action a and observe o. In POMDP formulation, o is the observation which is in the state space of the original MDP, thus we can rewrite Eq.(3) as

$$V^{t+1}_s(b) = \max_a \sum_{s'} \Pr(s'|s,b,a)[R(s,a,s') + \gamma V^t_{s'}(b^{s,s'}_a)] \tag{4}$$

The optimal value function can be represented by a set Γ of α-functions, and each α-function is a multivariate polynomial. For more detail, assuming that the optimal value function is a piecewise-linear convex function and written as

$$\begin{aligned} V^{t+1}_s(b) &= \max_{\alpha \in \Gamma} \int_\theta \alpha_s(\theta) b(\theta) d\theta \\ &= \max_{\alpha \in \Gamma^{t+1}} \alpha_s(b) \end{aligned} \tag{5}$$

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

$$\alpha^{t+1}_{b,s}(\theta) = \sum_{s'} \Pr(s'|s,\theta,a)[R(s,a,s') + \gamma \alpha^{s,s'}_{b,a}(\theta)] \tag{6}$$

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

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.(6), then α^t is a multivariate polynomial, if α^{t-1} is a multivariate polynomial. This is the major result of BEETLE algorithm [4].

However, we observe that the number of monomials in this representation increase in a factor of $O(|\mathcal{S}|)$ (see Eq.(6)). Thus, this representation causes the main intractability in Bayesian RL. In the next section, we will introduce one approximation algorithm which has capability of efficiently dealing with this problem. We will implicitly represent α-functions as a finite state controller.

III. Partially Observable Monte-Carlo Planning (POMCP)

The POMCP algorithm [8] is an online planning method that extends Monte-Carlo tree search (MTCS) method [10] to POMDPs. Each node of the tree search is represented by a pair 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 the value of history h, 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 [11]. If child nodes for all actions of the current node are added into the tree, actions are selected how to maximize an upper confidence bound on the action value,

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

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

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 can be considered to 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.

IV. Monte-Carlo Planning for Bayesian RL

Partially Observable Monte-Carlo Planning (POMCP) was introduced in [8] for solving continuous or large POMDPs. It uses particle filter for belief representation and a UCT search method [10] to select actions online. Instead of using a particle filter, we represent the belief in a closed form using 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 and Bayesian Forward Search Sparse Sampling (BFS3) [12]; 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 [6] 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 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 [8]. Lemma 4.1 is a straightforward derivation from [8]. By this Lemma, the convergence analysis of MCBRL is equivalent to POCMP [8] and UCT [10].

Lemma 4.1: 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

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

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, $\tilde{V}^\pi(h) = V^\pi(h), \forall\pi$.

V. 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 [4]. We then use a mixture of model $p(\theta) = \sum_i^n \beta_i p(\theta|M_i)$ as the prior. 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.

Assume that the mixture model is

$$p(\theta) = \sum_i \beta_i p(\theta|M_i) \tag{7}$$

Algorithm 1 Monte-Carlo Online Planning Algorithm for Bayesian RL

procedure ONLINESEARCH(s, h)
1: **for** $i = 1$ to N **do**
2: Sample a model $\theta \sim B(h)$.
3: SIMULATE$(s, \theta, h, 0)$.
4: **end for**
5: **return** $\underset{a}{\text{argmax}}\ Q(s, h, a)$.

procedure SIMULATE(s, θ, h, d)
1: **if** $(\gamma^d < \epsilon)$ **then**
2: return 0
3: **end if**
4: **if** $(h \notin T)$ **then**
5: **for** all $a \in \mathcal{A}$ **do**
6: Add a node (s, θ, ha, d) into the tree T.
7: **end for**
8: **return** ROLLOUT(s, θ, h, d)
9: **end if**
10: Select an action $a \leftarrow \underset{b}{\text{argmax}}\ \{Q(hb) + c\sqrt{\frac{\log N(h)}{N(hb)}}\}$.
11: Take action a, and observe $(o, r) \sim \theta$.
12: $R \leftarrow r + \gamma$.SIMULATE$(o, \theta, hao, d + 1)$
13: $N(h) \leftarrow N(h) + 1$, $N(ha) \leftarrow N(ha) + 1$
14: $Q(ha) \leftarrow Q(ha) + \frac{R - Q(ha)}{N(ha)}$
15: **return** R

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

where π 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

$$p(\theta|\text{data}) = \sum_i p(M_i|\text{data})p(\theta|M_i, \text{data}). \tag{8}$$

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

$$p(M_i|\text{data}) \propto \beta_i p(\text{data}|M_i) \tag{9}$$

we need compute $p(\text{data}|M_i)$ as

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

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

$$p(\text{data}|M_i) = \prod_{c,a} \frac{\Gamma(\sum_j \gamma_j)}{\prod_j \Gamma(\gamma_j)} \frac{\prod_s \Gamma(\sum_j o_j^{s,a} + 1)}{\prod_{j,s} \Gamma(o_j^{s,a} + 1)} \frac{\prod_j \Gamma(\sum_s o_j^{s,a} + \gamma_j)}{\Gamma(\sum_{j,s} o_j^{s,a} + \gamma_j)}$$

(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 (8). Thus, in order to apply this nested mixture model for 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 (9), then sample the dynamics from $\theta_{s,a}^{s'} \sim p(\theta|M_i, \text{data})$ as before in Section IV.

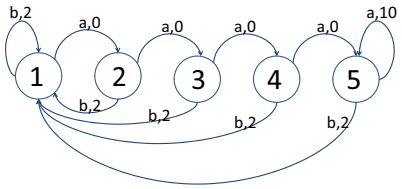

Figure 1. Chain problem showing action, reward for each transition.

VI. Experiments

In this section, we evaluate the performance of MCBRL on two simple problems from the previous works: "Chain" [2], [4], [13] and 6×6 maze [13], [14], and compare with BEETLE [4], BOSS [13], and BFS3 [15] algorithms.

A. Chain Problem

Fig.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 *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 [8], 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.

Table I shows the results of MCBRL compared to other algorithms from chain problem. The default policy is the *Exploit* policy which is the MDP policy for the average model from the current belief [4]. 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

Table I
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

Semi setting. Moreover, MCBRL can be also effective in Full setting in which BEETLE fails to do more carefully exploration due to large parameter space. However, MCBRL still does exploration less effective than BOSS and BFS3 in large parameter space. We have tried the *Exploit* policy [16] to replace the rollout policy (with informed rollout policy), but the performance even decreases. The poorer performance of MCBRL with informed rollout policy is consistent with the discovery in [16].

We also test MCBRL with the use of particle filter which does not need closed form representations of belief. The model belief is updated approximately using an unweighted particle filter, which is similar to the POMCP algorithm [8]. 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 Section V, we use a variant of the chain problem as in [13], named Chain2. Chain2 problem includes one more cluster. 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. For summary, we put the results of two problems in Tables II. The results show that MCBRL with nested mixture models can quickly find and learn the correct tied model as described in Fig. 2.

Table II
RESULTS FOR CHAIN & CHAIN2

	Chain	Chain2
Tied	3653 ± 32	1673 ± 34
Semi	3650 ± 34	3241 ± 34
Full	2675 ± 53	2475 ± 41
Mixture	3650 ± 39	3164 ± 45

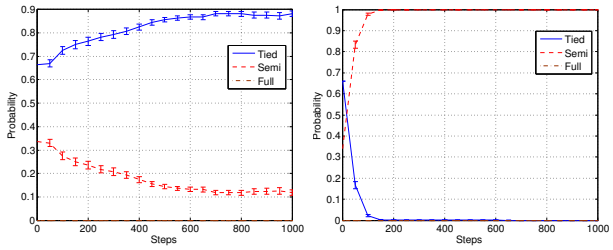

Figure 2. The probabilities of each tied model in (left) Chain and (right) Chain2 problems. The error bars are standard deviations.

B. Maze Problem

In this section, we experiment with the 6×6 maze problem as in [13] and [14]. 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% of intended effect, and 20% noisy effect which makes the agent to 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 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 models of those tying models.

The cumulative rewards obtained by BOSS [13], BFS3 [15], MCBRL with three separate models, and MCBRL with the nested mixture model (Section V) are shown in Fig.4. Our MCBRL algorithm with the nested mixture model performs better than BOSS and BFS3 (with nonparametric model clustering) which is able to find only about 10 clusters. This is because the MCBRL algorithm uses a fixed

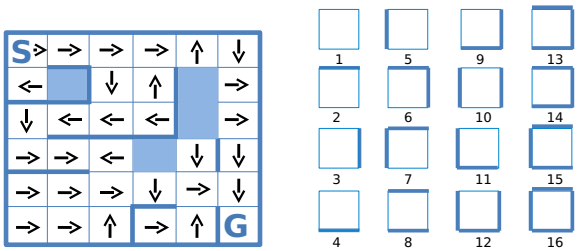

Figure 3. 6×6 maze problem with an optimal policy (on the left) and 16 clusters (on the right).

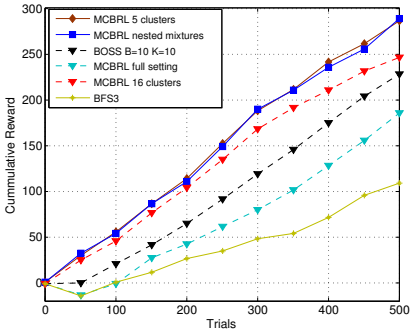

Figure 4. Cumulative training reward vs. trials for 6×6 maze.

set of tying models which makes it quickly find and learn the best one (in term of capturing the problem's correct dynamics) with only 5 clusters. BFS3 gives a poorer result because it is a bread-first search algorithm, so the number of expanded nodes in tree search increases exponentially with the depth of the tree. In order to find the goal state in this Maze problem, the average number of steps of the optimal policy is about 90 steps. Thus, the depth of the tree search is often set larger than that number. Unfortunately, our computer does have enough memory to construct such a large tree if BFS3 is used.

VII. 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 value function representation with time horizon. We further propose the use of 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 increase 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.

References

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

[2] R. Dearden, N. Friedman, and S. J. Russell, "Bayesian Q-learning," in *AAAI*, 1998, pp. 761–768.

[3] M. Ghavamzadeh and Y. Engel, "Bayesian policy gradient algorithms," in *NIPS*, 2006, pp. 457–464.

[4] P. Poupart, N. A. Vlassis, J. Hoey, and K. Regan, "An analytic solution to discrete bayesian reinforcement learning," in *ICML*, 2006, pp. 697–704.

[5] N. A. Vien, H. Yu, and T. Chung, "Hessian matrix distribution for bayesian policy gradient reinforcement learning," *Inf. Sci.*, vol. 181, no. 9, pp. 1671–1685, 2011.

[6] T. Wang, D. J. Lizotte, M. H. Bowling, and D. Schuurmans, "Bayesian sparse sampling for on-line reward optimization," in *ICML*, 2005, pp. 956–963.

[7] M. Duff, "Optimal learning: Computational procedures for bayes-adaptive markov decision processes," PhD Thesis, University of Massassachusetts Amherst, 2002.

[8] D. Silver and J. Veness, "Monte-carlo planning in large POMDPs," in *NIPS*, 2010, pp. 2164–2172.

[9] D. Hsu, W. S. Lee, and N. Rong, "What makes some POMDP problems easy to approximate?" in *NIPS*, 2007.

[10] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *ECML*, 2006, pp. 282–293.

[11] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[12] T. J. Walsh, S. Goschin, and M. L. Littman, "Integrating sample-based planning and model-based reinforcement learning," in *AAAI*, 2010.

[13] J. Asmuth, L. Li, M. L. Littman, A. Nouri, and D. Wingate, "A Bayesian sampling approach to exploration in reinforcement learning," in *UAI*, 2009.

[14] S. J. Russell and P. Norvig, *Artificial Intelligence: A modern approach*, 2nd ed. Upper Saddle River, N.J.: Prentice Hall, 2003.

[15] J. Asmuth and M. L. Littman, "Learning is planning: near Bayes-optimal reinforcement learning via Monte-Carlo tree search," in *UAI*, 2011, pp. 19–26.

[16] S. Gelly and D. Silver, "Combining online and offline knowledge in uct," in *ICML*, 2007, pp. 273–280.