

---

# Improving Bayesian Reinforcement Learning Using Transition Abstraction

---

**Daniel Acuna**

Department of Computer Science & Engineering  
University of Minnesota

ACUNA002@UMN.EDU

**Paul Schrater**

Departments of Computer Science & Engineering and Psychology  
University of Minnesota

SCHRATER@UMN.EDU

## Abstract

Bayesian Reinforcement Learning (BRL) provides an optimal solution to on-line learning while acting, but it is computationally intractable for all but the simplest problems: at each decision time, an agent should weigh all possible courses of action by beliefs about future outcomes constructed over long time horizons. To improve tractability, previous research has focused on sparsely sampling possible courses of action that are most relevant to computing value; however, sampling alone does not scale well to larger environments. In this paper, we investigate whether an abstraction called projects—parts of the transition dynamics that bias the look ahead to areas of the environment that are promising—can scale up BRL to larger environments. We modify a sparse sampler to incorporate projects. We test our algorithm on standard problems that require effective exploration–exploitation balance and show that learning can be significantly sped up compared to a simpler BRL and the classic Q-learning.

**Keywords:** Bayesian reinforcement learning, Sparse Sampling, Transition Abstraction

## 1. Introduction

Bayesian Reinforcement learning (BRL) is a model-based approach to Reinforcement Learning (RL) that provides a principled solution to the problem of simultaneous planning and learning under uncertainty, including partially observed states and unknown or variable state dynamics. The theoretical foundations of BRL methods originate in the work of Bellman (Bellman, 1957; Bellman, 1961), Howard (Howard, 1960), and Feldhaum (Fel'dhaum, 1965) and others in Control Theory and Operations Research, but had long been considered computationally intractable outside of a few special cases. While in theory, traditional RL can be used to find an optimal policy for problems with partially observed states and unknown transition matrices, in the on-line setting, RL methods suffer from slow convergence, require too much data, and must implement methods to handle the exploration/exploitation trade-off. In contrast, BRL is better suited to on-line learning because it can use the prior knowledge embedded in the model to speed-up convergence, decrease the amount of data required, and it can optimally trade-off exploration/exploitation.

In general, BRL remains computationally intractable, with action selection forming the critical limitation. The difficulty is more apparent when BRL is cast as a belief MDP or a POMDP. In this form, policy learning involves estimating a value function over possible belief states, which exponentially explodes with increases in the future horizon. Recent efforts to find good approximations for BRL have generated a variety of approaches (Castro & Precup, 2007; Wang et al., 2005; Kearns et al., 2002; Kearns & Singh, 2002; Strens, 2000). All these approaches can be construed as more efficient ways to compute the expected value of an ac-

---

Appearing in Proceedings of the ICML/UAI/COLT Workshop on Abstraction in Reinforcement Learning, Montreal, Canada, 2009. Copyright 2009 by the author(s)/owner(s).

tion via approximations.

The goal of this paper is answer whether or not transition abstraction can speed up BRL. To achieve this, we propose an sparse sampler that can incorporate projects—structural assumptions about a task that can be used to bias planning. In section 2, we describe BRL and the terminology used throughout the paper. In section 3, we describe the algorithm. In section 4, we show that our algorithm is able to use the projects to speed up learning and outperform BRL without projects and classic Q-learning.

## 2. Background: Bayesian Reinforcement Learning

Bayesian Reinforcement Learning (BRL) theory originated in early work in control and operations research (i.g., see (Bellman, 1961; Howard, 1960; Bellman, 1957; Fel'dhaum, 1965)). BRL extends Markov Decision Processes (MDP) by the addition of probability models that capture transition uncertainty. Several formulations exist as Partially-Observable MDP (POMDP), or Bayesian-Adaptive MDP, Meta-MDP and Belief MDP (Wang et al., 2005; Duff, 2002). We use the POMDP formulation (Poupart et al., 2006; Duff, 2002) because of its relative simplicity (of language) and elegance.

An MDP is a stochastic rewarding process with control defined by  $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ , where  $\mathcal{S}$  is a set of states,  $\mathcal{A}$  is a set of actions,  $T(s, a, s') \in \mathcal{T}$  is a transition probability function defining the conditional probability  $Pr(s'|s, a)$  of going to state  $s'$  by taking action  $a$  at state  $s$ , and  $R(s, a, s') \in \mathcal{R}$  is the reward gathered when state  $s'$  is reached after taking action  $a$  at state  $s$ . A policy  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  governs (possibly stochastically) the action to take at each state. The aim of the agent is to find a policy that maximizes the total discounted reward  $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$ , where  $0 < \gamma < 1$  is a discount factor and  $r_t$  is the reward gathered at time  $t$ .

Reinforcement learning (RL) comprises finding the optimal policy for an MDP whose transitions and/or rewards are unknown. Thus, RL needs to actively interact with the environment whereas MDP does not. In this setting, a natural conflict appears, namely balancing the need to maximize learning (explore) and maximize reward (exploiting) while acting. BRL proceeds by putting a probability model over uncertainties. Each unknown transition probability  $T(s, a, s')$  is an unknown parameter  $\theta_a^{s, s'}$ .

The POMDP formulation (Poupart et al., 2006; Duff, 2002; Kaelbling et al., 1998) is described by a tuple

$\langle \mathcal{S}_P, \mathcal{A}_P, \mathcal{O}_P, T_P, Z_P, R_P \rangle$ . The set  $\mathcal{S}_P = \mathcal{S} \times \left\{ \theta_a^{s, s'} \right\}$  is a cross product of the “core” MDP states  $s \in \mathcal{S}$  with parameters  $\theta_a^{s, s'}$ . States that belong to  $\mathcal{S}_P$  are sometimes called hyper-states. The action space  $\mathcal{A}_P = \mathcal{A}$  is the same as the core MDP. For simplicity, we restrict the discussion to perfect observation of core states ( $\mathcal{O}_P = \mathcal{S}$ ). We factor the transition function  $T_P(s, \theta, a, s', \theta') = Pr(s', \theta' | s, \theta, a)$  into two conditional distributions, one for MDP states and one for unknown parameters:  $Pr(s' | s, \theta^{s, s'}, a) = \theta_a^{s, s'}$  and  $Pr(\theta' | \theta) = \delta_\theta(\theta')$ , respectively, where  $\delta_\theta(\theta')$  is a Kronecker delta with value 1 when  $\theta' = \theta$  and 0 otherwise.  $Pr(\theta' | \theta) = \delta_\theta(\theta')$  reflects a stationary environment assumption. The observation function  $Z_P(s', \theta', a, o) = Pr(o | s', \theta', a)$  indicates the probability of making observation  $o$  when state  $s', \theta'$  is reached after executing action  $a$ . Assuming perfect observation,  $Pr(o | s', \theta', a) = \delta_{s'}(o)$ . The reward function  $R_P(s, \theta, a, s', \theta') = R(s, a, s')$  is the same as the core MDP since it does not depend on  $\theta$  or  $\theta'$ .

Using the POMDP formulation, we can learn the transition model  $\theta$  by belief updating. After observing  $s, a, s'$ , the belief  $b_t(\theta) = Pr(\theta)$  at time  $t$  over parameters  $\theta_a^{s, s'}$  is updated using Bayes' rule

$$\begin{aligned} b_t(\theta) &\propto b_{t-1}(\theta) Pr(s' | \theta, s, a) \\ &\propto b_{t-1}(\theta) \theta_a^{s, s'} \end{aligned} \quad (1)$$

In POMDP, a policy  $\pi$  is a mapping from belief states to actions (i.e.,  $\pi(b) = a$ ). The value of a belief state  $b$  under a policy  $\pi$  is the expected discounted total reward  $V^\pi(b) = \sum_{t=0}^{\infty} \gamma^t R(b_t, \pi(b_t), b_{t+1})$ . An optimal policy  $\pi^*$  has the highest value in all belief states (i.e.,  $V^{\pi^*}(b) \geq V^\pi(b), \forall \pi, b$ ) and its value function can be computed by Bellman's equations (Bellman, 1957). Within BRL, the Bellman's equation can be re-written as

$$V_s^*(b) = \max_{a \in \mathcal{A}} \sum_{s'} Pr(s' | s, b, a) \left( R(s, a, s') + \gamma V_{s'}^*(b'_a) \right) \quad (2)$$

where  $b$  is the current belief in  $\theta$  and  $b'_a$  is the updated belief following the optimal policy (Poupart et al., 2006).

### 2.1. Action selection by sparse sampling

Given the intractability of optimal Bayesian reinforcement learning, several techniques have been proposed to sparsely sample future actions to more effectively estimate values (Dearden et al., 1998; Dearden et al., 1999; Duff, 2002; Strens, 2000; Wang et al., 2005; Castro & Precup, 2007). We use a method based on linear programming that has shown to work well in prac-

**Algorithm 1** Action selection

INPUT:  $\{\psi_j\}_M =$  projects,  $H =$  horizon of general hyper-space tree,  $H_M =$  horizon projects hyper-tree,  $s =$  start state,  $\mathcal{S}_P$  states,  $\mathcal{A}$ : actions, *samples*: maximum number of samples,  $Q$ : state-value function

1. While (number of nodes in the hyper-tree is less than *samples*)
  - (a) Sample a trajectory of length  $H$  using as a root node  $s \times b$  considering that:
    - i. If during the trajectory a project  $j$  is found, we expand a tree of depth  $H_M$  of future outcomes using the project's belief  $b_j$ ;
    - ii. If a project ends during the sampled trajectory, continue the sampling using the full belief  $b$  and general trajectory length  $H$ ;
2. For each leaf of the hyper-space tree, use the maximum  $Q$ -value stored as its value
3. Compute the values of the hyper-space tree using linear programming
4. Select the action  $a$  that leads to the highest value hyper tree and observe next state  $s'$  and reward  $r$
5. Update  $Q$  according to the observed next state  $s'$ , reward  $r'$  and current state  $s$
6. Update beliefs with transition  $(s, a) \rightarrow s'$

tice (Castro & Precup, 2007). In essence, the method samples a set of trajectories from the hyper-MDP up till a fixed horizon and maintains an estimate of the state-action values associated to a set of core states. These stored state-action values are used to fill in the values of missing subtrees of the sampled hyper-tree. Once the hyper-tree is built, the state-action values are solved by linear programming. And finally, the old state-action value estimates are updated by the values computed by linear programming.

### 3. Adding Projects to Bayesian Reinforcement Learning

The intuition behind projects comes from Multi-armed Bandit Problems (MAB) (Gittins, 1989), which have an elegant structure for high-level planning: projects (arms) are independent, and the planning problem separates into scheduling projects and learning them. This is possible because projects partition beliefs; planning within a project does not change beliefs about any other project. In the general BRL, this is a strong assumption that, if approximately correct, can help speed up learning and action selection as well.

Projects provide an early bias to discard unlikely tran-

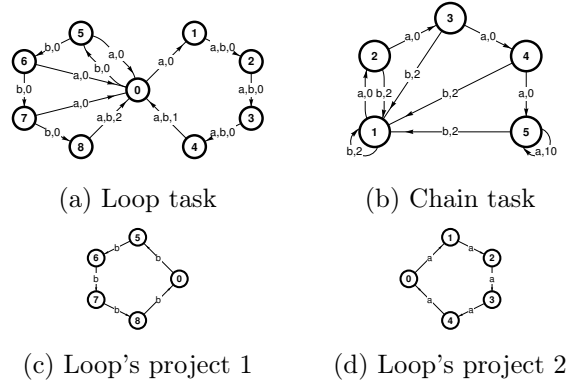


Figure 1. Test problems

sitions. Once the agent is engaged in a project, it assumes a null probability of transitioning to another project. In contrast, a BRL agent that does not use projects would consider all possible states as candidates for transition, which will generate unnecessary samples for futures that are very unlikely and low in value.

To introduce notation, suppose there are  $M$  projects  $\psi_1, \dots, \psi_M$  defined for a task, where a project  $\psi$  is a set of transition tuples  $(s, a, s')$ . A transition tuple belongs to only one project ( $\forall s, a, s' (s, a, s') \in \psi_j \wedge (s, a, s') \in \psi_k \Rightarrow j = k$ ). If a transition tuple does not belong to any of the defined project, then it is part of the residual transitions  $\psi_0$  defined by  $\{(s, a, s') : (s, a, s') \notin \psi_j, \text{ for } j = 1, \dots, M\}$ .

Projects essentially impose a partition on beliefs:

$$b(\{\theta_{ss'}^a\}) = \sum_{j=0,1,\dots,M} b_j(\{\theta_{ss'}^a : (s, a, s') \in \psi_j\}), \quad (3)$$

where we call  $b_0$  the residual belief, and  $b$  the full belief. Notice that in the usual BRL formulation, the residual belief is equal to the full belief. Given a project's belief, the predictive probability of a transition that is not part of the project has zero probability; this is,  $p(s' | s, a, b_j) = 0$  for all  $(s, a, s') \notin \psi_j$ .

Our algorithm (Algorithm 1) is a simple modification of a general sampler. We exploit the structure given by projects, strongly biasing how the hyper-space tree of possible futures grows. We start using the full belief (line 1.a of the Algorithm), but once a sampled transition belongs to a project (line 1.a.i), the algorithm enters a sampling *trance* where only the project's belief are used. While in this trance, if the sampling cannot continue because there are no transitions from the current core state forward (line 1.a.ii), the trance

ends and the sampling continues using the full belief again.

The tractability of the procedure is possible by a number of computational constraints. First, the algorithm restricts the maximum depth of the hyper-space tree, which corresponds to the look ahead horizon. However, we permit a larger maximum depth for projects to increase the accuracy of the value computation to focus the computation on the important regions of the hypertree. Finally, we restricted the total number of samples.

After the sampling stops, the leaves of the hyper-space tree are associated with core state values using Q values from previous computations (line 2). The algorithm then computes the value of actions (line 3), updates the Q-value of the current core state using the classical temporal difference algorithm (line 5) (Watkins, 1989) and updates the beliefs (line 6).

#### 4. Experimental Results

We implemented and tested our algorithm on the following two standard problems (Figure 1):

**Loop** This problem consists of two 5-state loops, which are connected at a single start state. Two deterministic actions are available. Since taking action  $b$  at every state in the left loop yields a reward of 2, the optimal policy is to perform action  $b$  everywhere and would yield a total reward of 100 after 250 steps. To attain the optimal policy, the agent needs to find the middle ground between exploration and exploitation, but it is difficult because the agent can get trapped in the right loop and obtain a series of smaller rewards.

**Chain** The chain problem consists of five states. There are two actions  $a$  and  $b$  available for the agent, but, with probability 0.2, the action will have resulted in an opposite effect. The optimal policy for this problem is to perform action  $a$  at every state and would generate a total reward of 919 after 250 steps. The difficulty is that the agent can be misled by the immediate rewards from taking action  $b$ . Therefore, the agent needs to explore effectively and estimate the discounted reward accurately.

In our tests, we assume the reward function is known. We test our algorithm with no projects, with 1 project, and with 2 projects. For comparison, we show the Q-learning performance with  $\epsilon$ -greedy action selection strategy. On both tasks, the hand-coded projects are

intended to bias planning towards areas that are most relevant to exploration-exploitation balance. The first project bias planning towards harder to learn but higher rewarding areas; the second project bias planning towards easier to learn but lower rewarding areas. By learning these two projects, the agent rapidly benefits from these biases by distinguishing which of the two rewarding areas is best.

On the loop environment, project 1 (Figure 1c) consists of the transition of action  $b$  traversing the left hand side loop; this is,  $\psi_1 = \{(0, b, 5), (5, b, 6), (6, b, 7), (7, b, 8), (8, b, 0)\}$ . Project 2 (Figure 1d) consists of the transition of action  $a$  traversing the right hand side loop; this is,  $\psi_2 = \{(0, a, 1), (1, a, 2), (2, a, 3), (3, a, 4), (4, a, 0)\}$ . In the chain environment, the first project contains the transitions of action  $a$  that take the agent from state 1 to state 5; this is,  $\psi_1 = \{(1, a, 2), (2, a, 3), (3, a, 4), (4, a, 5), (5, a, 5)\}$ . The second project contains the self-transition of state 1 by taking action  $b$ ; this is,  $\psi_2 = \{(1, b, 1)\}$ . For the Chain and Loop environments, we use a maximum of 150 samples, a horizon of 3 for the general tree, and a horizon of 5 the projects. For the Q-learning comparison, we found the best performance with  $\epsilon = 0.05$  and  $\alpha = 0.3$ . In Figure 2, we show how much reward each algorithm accumulates every 250 steps for a total of 6000 steps. For each algorithm, we average the performance over 100 runs.

The no-project BRL gets an important boost in reward gathering if we add the hand-coded projects. Project 1 adds a boost for reward maximization. With the second project, the performance is improved further because it adds an additional boost for *learning*—even though project 2 will not lead to the higher rewarding areas, it will allow the agent to compare projects that seem to offer competing reward schedules.

#### 5. Conclusions

In this work, we show that adding transition abstraction to Bayesian Reinforcement Learning can speed up learning. Our tests on standard problems show how meaningful projects may add significant improvements compared to standard BRL and Q-learning. The results suggests that projects do not need to bias toward high rewards, but rather be useful for learning competitive courses of actions. In the future, our hope is to incorporate automatic discovery and/or hierarchization of projects, which is closely related to current research in automatic discovery of options, hierarchies, or factorization (Barto & Mahadevan, 2003).

References

Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13, 41–77.

Bellman, R. E. (1957). *Dynamic programming*. Princeton: Princeton University Press.

Bellman, R. E. (1961). *Adaptive control processes: A guided tour*. Princeton University Press.

Castro, P. S., & Precup, D. (2007). Using linear programming for bayesian exploration in markov decision processes. *IJCAI* (pp. 2437–2442).

Dearden, R., Friedman, N., & Andre, D. (1999). Model based bayesian exploration. *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence* (pp. 150–159).

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

Duff, M. O. (2002). *Optimal learning: Computational procedures for bayes-adaptive markov decision processes*. Doctoral dissertation, University of Massachusetts Amherst.

Fel'dbaum, A. (1965). *Optimal control systems*. Academic Press.

Gittins, J. C. (1989). *Multi-armed bandit allocation indices*. Chichester [West Sussex] ; New York: Wiley.

Howard, R. (1960). *Dynamic programming*. Cambridge, MA: MIT Press.

Kaelbling, L., Littman, M., & Cassandra, A. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 99–134 123 124 125 1.

Kearns, M., Mansour, Y., & Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49, 193–208.

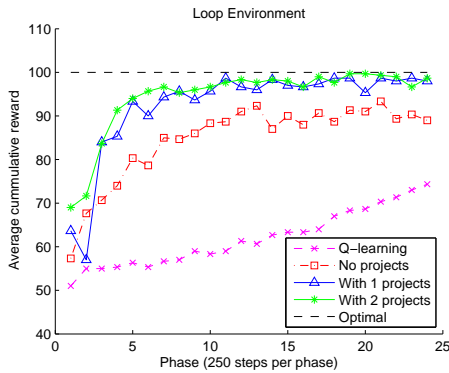
Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49, 209–232.

Poupart, P., Vlassis, N., Hoey, J., & Regan, K. (2006). An analytic solution to discrete bayesian reinforcement learning. *23rd International Conference on Machine Learning*.

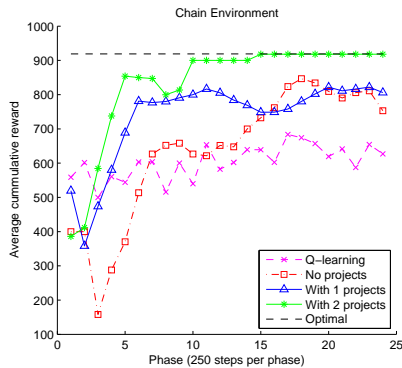
Strens, M. J. A. (2000). A bayesian framework for reinforcement learning. *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 943–950). Morgan Kaufmann Publishers Inc.

Wang, T., Lizotte, D., Bowling, M., & Schuurmans, D. (2005). Bayesian sparse sampling for on-line reward optimization. *International Conference on Machine Learning*. Bonn, Germany.

Watkins, C. (1989). *Learning from delayed rewards*. Doctoral dissertation, University of Cambridge.



(a) Loop environment



(b) Chain environment

Figure 2. Results in Chain environment