Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Efficient Learning in Linearly Solvable MDP Models

Ang Li
Department of Computer Science,
University of Minnesota USA
lixx1522@umn.edu

Paul R Schrater
Department of Computer Science & Psychology,
University of Minnesota USA
schrater@umn.edu

Abstract

Linearly solvable Markov Decision Process (MDP) models are a powerful subclass of problems with a simple structure that allow the policy to be written directly in terms of the uncontrolled (passive) dynamics of the environment and the goals of the agent. However, there have been no learning algorithms for this class of models. In this research, we develop a robust learning approach to linearly solvable MDPs. To exploit the simple solution for general problems, we show how to construct passive dynamics from any transition matrix, use Bayesian updating to estimate the model parameters and apply approximate and efficient Bayesian exploration to speed learning. In addition, we reduce the computational cost of learning using intermittent Bayesian updating and policy solving. We also gave a polynomial theoretical time complexity bound for the convergence of our learning algorithm, and demonstrate a linear bound for the subclass of the reinforcement learning problems with the property that the transition error depends only on the agent itself. Test results for our algorithm in a grid world are presented, comparing our algorithm with the BEB algorithm. The results showed that our algorithm learned more than the BEB algorithm without losing convergence speed, so that the advantage of our algorithm increased as the environment got more complex. We also showed that our algorithm's performance is more stable after convergence. Finally, we show how to apply our approach to the Cellular Telephones problem by defining the passive dynamics.

1 Introduction

Markov Decision Processes (MDP) provide the foundation for key problems in artificial intelligence, including robotics, automated decision making and scheduling. The problem of learning MDPs (Reinforcement learning) is challenging and has given rise to a broad assortment of approaches that vary in terms of how much must be known about the underlying MDP, ranging from model-free methods that require almost no information, (e.g. adaptive heuristic critic algorithms and Q-learning) [Kaelbling *et al.*, 1996] to model-based algorithms that need the MDP to be partially specified (e.g. the Dyna algorithm, the Queue-Dyna algorithm and the RTDP algorithm) [Kaelbling *et al.*, 1996]. Given limited data and some background knowledge about the underlying MDP, Model-based algorithms allow for more rapid learning by exploiting structure of the learning process and solving the exploration problem. A critical problem in using model-based algorithms is the computational cost–to do learning the Bellman equation [Bellman, 2003] defined by the MDP must be repeatedly solved for many different versions of the model.

The computational cost can be reduced when the Bellman equation has analytic solutions. Recently, Todorov [Todorov, 2009] described a class of MDPs with linear solutions, and showed that most discrete control problems can be approximated by this class. Critically, policies are analytically computed using only the passive dynamics and the reward function. In this paper, we focus on combining model-based learning with Todorov's result, incorporating an exploration strategy to trade-off exploration and exploitation.

Computational cost can be further reduced by delaying Bayesian updating of MDP model parameters to reduce the need for recomputing the policy. However, the convergence speed of the policy critically depends on updating the model with new observations. Thus, how to delay updating and how long it can be delayed become crucial. In our research we describe a function to control the updating process and give a time bound for convergence using this approach.

1.1 Problem Description

MDP models

The MDP model is a stochastic rewarding process with control defined by a 4-tuple (S, A, R, T), where S is a set of states, A is a set of actions, R is the expected reward gathered after taking action a at state s and $T : S \times A \times S \to [0, 1]$ is a transition probability function defining the conditional probability $p(s_t|s_{t-1}, a_{t-1})$ of going to state s_t by taking action a_{t-1} at state s_{t-1}. A policy $\pi : S \to A$ is a function that maps states into actions, guiding which action to take at each state. The aim of the algorithm is to find a policy π^* that maximizes the total expected discounted reward $r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots$, where r_i is the reward gathered at time t and γ is a discount factor. This problem can be solved

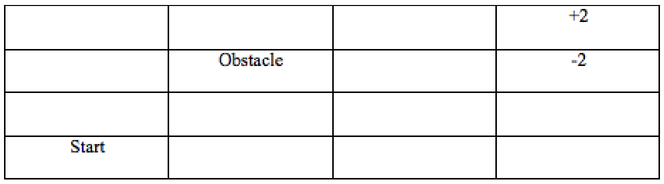

Figure 1: A 4×4 grid world

by solving a set of Bellman equations [Bellman, 2003]

$$V_T^*(s_t) = \max_{a_t \in A}\{R(s_t, a_t) + \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t)V^*(s_{t+1})\} \tag{1}$$

and selecting the action inside the max operator as the optimal policy [Puterman, 1994]. In RL, neither the transition function nor the reward function is assumed precisely known. For this research the reward function was assumed known with the focus on learning T.

Example Learning Problem

Suppose that an agent lives in an $m \times m$ grid world (4×4 is shown in Figure 1). Beginning in the starting state (lower left corner entry), it must choose an action at each time step. The interaction with the environment terminates when the agent reaches one of the goal states (upper right corner entry and the entry under it, marked $+2$ or -2). The agent gets the reward $+2$ or -2 at the goal state and -0.04 at all other states. In each location, the available actions are Up, Down, Left and Right. It is assumed that the environment is fully observable so that the agent always knows where it is. The states, except the start state and the terminal states, can have obstacles associated with them with a probability of 0.1 ($p = 0.1$), that prevent the agent from entering. The environment has a transition error: after an action, the intended outcome occurs with $p = 0.8$, and the agent moves perpendicularly to the intended direction with $p = 0.2$. However, this transition error is unknown by the agent. The task of the agent is to learn this environment, getting the optimal policy at each state that maximizes the overall reward while learning the transition error.

2 Related Work

2.1 Bayesian Reinforcement Learning

Strens [Strens, 2000] proposed a Bayesian solution for model-based RL, which explicitly maintains a *belief* over models. *Belief* is denoted by B, where $B = P(T)$. After making an observation, B is updated using Bayes' rule, $B_{t+1}|s_t, a_t, s_{t+1} \propto p(s_t, a_t, s_{t+1}|T) \times B_t$. In this algorithm B is maintained by a set of Dirichlet distributions over T [Teh *et al.*, 2006; Pitman and Yor, 1997]. Incorporating B and augmenting the state to the tuple (b, s), the Bellman equation becomes

$$\begin{aligned} V^*(b_t, s_t) = \quad & \max_{a_t \in A}\{R(s_t, a_t) + \qquad (2) \\ + \quad & \gamma \sum_{s_{t+1}} p(s_{t+1}|b_t, s_t, a_t)V^*(b_{t+1}, s_{t+1})\} \end{aligned}$$

The RL process can be described as the iteration of 1) interacting with the environment, getting an observation, 2) updating B and 3) solving the Bellman equation given B.

2.2 Near-Bayesian Exploration in Polynomial Time

In general, solving the set of equations derived from equation 2 is extremely hard since the optimal action is based not only on how the action will affect the next state but also how the action will affect the next B (i.e., equation 2 depends on b and b' rather than just b) and approximations must be done. Kolter and Ng [Kolter and Ng, 2009] proposed an approximation algorithm called the Bayesian Exploration Bonus Algorithm (BEB) which chooses actions according to the current mean estimate of the state transition plus an additional reward bonus for state-action pairs. According to their algorithm, the Bellman equation was redefined as

$$\begin{aligned} V^*(b_t, s_t) = \quad & \max_{a_t \in A}\{R(s_t, a_t) + \frac{\beta}{1 + \alpha_0(s_t, a_t)} + \quad (3) \\ + \quad & \gamma \sum_{s_{t+1}} p(s_{t+1}|b_t, s_t, a_t)V^*(b_t, s_{t+1})\} \end{aligned}$$

where β is a constant and $\alpha_0(s_t, a_t)$ is the count of observations of state-action pair (s_t, a_t). Note that B is not updated in this equation (i.e., it only depends on b rather than b and b'), thus the equation can be solved using standard value iteration or policy iteration algorithms.

2.3 Efficient Computation of Optimal Actions

While simpler, the model-based computation of optimal actions in equation 3 is usually intractable and typically requires extensive approximation. However, efficient and exact computation of optimal actions is possible for a subclass of MDP models [Todorov, 2009]. Consider now the cost instead of the reward in equation 1; $l(s, a)$ is the cost at state s when taking action a, and $l(s, a) = -R(s, a)$. The problem then becomes minimizing the total cost instead of maximizing the total reward, and the Bellman equation is rewritten as

$$V(s) = \min_a\{l(s, a) + E_{s' \sim p(\cdot|s,a)}[V(s')]\} \tag{4}$$

where $E[V(s')]$ is the expected cost-to-go at the next state. Todorov relaxed the optimization problem by allowing the agent to specify the transition probabilities $u(s'|s)$ directly instead of specify sequences of symbolic actions. Thus actions become $u(s'|s) = p(s'|s, a)$ and the agent has the power to reshape the dynamics in any way it wishes, given action costs. Let $uc(s'|s)$ denote the passive dynamics characterizing the behavior of the system in the absence of controls. The action costs can now be quantified by measuring the difference between $uc(s'|s)$ and $u(s'|s)$. Differences between probability distributions are usually measured by the Kullback-Leibler (KL) divergence, so the cost $l(s, a)$ must have the form

$$l(s, a) \quad = \quad q(s) + KL(u(\cdot|s) \parallel uc(\cdot|s)) \tag{5}$$

where $q(s)$ is the state cost, which can be an arbitrary function encoding how desirable different states are. Let $z(s) = \exp(-V(s))$ be the desirability at each state. Substituting equation 4 in equation 5, the Bellman equation can be written in terms of z as

$$-\log z(s) = q(s) + \min_{a}\{E_{s'\sim u(\cdot|s)}[\log \frac{u(s'|s)}{uc(s'|s)z(s')}\} \quad (6)$$

Since the KL divergence achieves its global minimum of zero when the 2 distributions are equal, the optimal action

$$\pi(s'|s) = \frac{uc(s'|s)z(s')}{g[z](s)}$$

where $g[z](s) = \sum_{s'} uc(s'|s)z(s') = E_{s'\sim uc(\cdot|s)}[z(s')]$, is the normalization factor. So the Bellman equation can now be simplified by substituting the optimal action, taking into account the normalization term, and exponentiating. Thus,

$$z(s) = \exp(-q(x))g[z](s) \quad (7)$$

If we define the index sets T and N, terminal and nonterminal states, and partition z, q, and uc accordingly, equation 7 becomes

$$(diag(\exp(q_N)) - uc_{NN})z_N = uc_{NT}\exp(-q_T) \text{ and}$$
$$z_N = (diag(\exp(q_N)) - uc_{NN})^{-1} \cdot uc_{NT}\exp(-q_T) \quad (8)$$

where z_N is the vector of desirability at the nonterminal state and the desirability at the terminal state is $V(s) = q(s)$, so $z(s) = \exp(-q(s))$. Now, the solution of the Bellman equation becomes a linear equation, and thus can be solved by matrix factorization. Most importantly, equation 8 only involves the passive dynamic and the reward function. Therefore, if the passive dynamic and the reward function are unchanged with the environment a linear Bellman equation need only be pre-solved once. However, Todorov's solution has some limitations. First, the control cost must be a KL divergence, which reduces to the familiar quadratic energy cost in continuous settings. This is a sensible way to measure control energy and is not particularly restrictive. Second, the controls and the noise must be able to cause the same state transitions. Todorov notes this is a more significant limitation, but for a discrete state MDP model this limitation is naturally satisfied. Third, the noise amplitude and the control costs are coupled. This can be compensated to some extent by increasing the state costs ensuring that $\exp(-q(x))$ does not become numerically zero.

3 Efficiently solving Bayesian RL using linearly solvable MDPs

In order to adequately explore the environment, ideally we would execute a full Bayesian look-ahead since the Bayesian solution (see section 2.1) automatically trades-off exploration and exploitation. Exploitation maximizes rewards using current estimates and exploration maximizes long-term rewards by improving estimates and they are both an indispensable part of the RL process. However, the Bayesian solution is intractable in general [Poupart, 2007]. In our approach, we use the approximation in equation 3 to balance exploration/exploitation. To efficiently solve this equation, our strategy is to exploit the linear solution given by equation 8. The next task is to combine Todorov's result with equation 3.

3.1 Passive Dynamics

First, to apply equation 8, we need to be able to define the uncontrolled dynamics. However, a simple method for constructing passive dynamics for almost every control problem exists. For example, an unbiased coin is the passive dynamic of a coin toss since this is tossing without control. The passive dynamic for the shortest-path problem on a graph corresponds to a random walk on the graph. For the problem at hand the passive dynamic is randomly choosing an action. Since there is a transition error it needs to be added to the passive dynamics, producing the equation

$$uc(s'|s) = \frac{\sum_{a=1}^{n_a} p(s'|s,a)}{n_a} \quad (9)$$

where n_a is the number of actions.

3.2 Choosing the Suboptimal Action from Desirability

Second, equation 8 solves a relaxed problem that computes the desired probabilities of transitioning to each state (the desirability). However, the agent still needs to select actions, not action probabilities. We solve this problem of choosing an action from desirability given the current *belief* state by projecting the desired probabilities onto an estimate of the transition probabilities to find the action that can best execute the desired probabilities. This produces the equation

$$a = \max_{a\in A}\{\sum_{s'} z(s')p(s'|s,a)\} \quad (10)$$

where a is the action and s is the current state.

3.3 The Learning Process

Similar to traditional Bayesian RL, we will maintain *belief* in the models. The variable *belief* is denoted as B, where $B = P(T)$. After making an observation, B is updated using Bayes' rule, $B_{t+1}|s_t, a_t, s_{t+1} \propto p(s_t, a_t, s_{t+1}|T) \times B_t$. In this algorithm, B about the transition distributions is maintained by a set of Dirichlet distributions over each T. Now we revise Todorov's solution by adding *belief*, defining the state by the tuple (b, s). Equation 9 now becomes

$$uc(s'|s) = \frac{\sum_{a=1}^{n_a} p(s'|s,b,a)}{n_a} \quad (11)$$

3.4 Exploration vs. Exploitation

To balance exploration and exploitation, similar to Kolter's method in Section 2.2, an exploration bonus was added when choosing the suboptimal action from equation 10, so equation 10 becomes

$$a = \max_{a\in A}\{\frac{\beta}{1+\alpha_0(s,a)} + \sum_{s'} z(s')p(s'|s,a)\} \quad (12)$$

where β is a constant and $\alpha_0(s,a)$ is the count of observations of the state-action pair (s,a). This procedure mirrors the approach to RL where adaptive dynamic programming is combined with an exploration function [Thrun, 1992]. However, after a number of iterations, b will never change (i.e., the environment is fully observed by the agent), so the exploration process needs to stop by removing the exploration bonus from equation 12. Between two RL rounds $(p(s'|s, b_{i-1}, a) - p(s'|s, b_i, a))^2$ is accumulated (set to be M in the algorithm in sections 4.3). An exploration switcher is initially set to ON and whenever the accumulated difference is smaller than some predetermined constant it is turned off.

Now the RL process can be described as the iteration of 1) interacting with the environment, producing an observation, 2) updating B, 3) solving equation 11 given B to get the passive dynamic, 4) solving equation 8 given the passive dynamic to get desirability, 5) Choosing the suboptimal action from equation 10 or 12 depending on the exploration switcher and 6) if it is the end of an RL round, checking the accumulated difference of this RL round to see whether the exploration switcher should be turned off.

4 Manual Delay of the Algorithm

4.1 Time Complexity and Delay of Updating

If there are n states total and h exploration steps are needed for convergence, then at each step equation 8 needs to be solved, which involves a matrix inverse operation of an n-by-n matrix that can be solved in $O(n^3)$ time ($O(n^{2.8})$ time with Strassen's algorithm). Therefore, the time cost is $O(h \times n^3)$. However, h might still be large. What is wanted is to separate h and n in time complexity as in the form $O(h + n^3)$. Note that some of the observations will change the suboptimal policy so little that solving the corresponding equations provide no benefit. Thus, doing Bayesian updates and solving equation 8 every time is inefficient. We want a control function that minimizes the total number of updates. Seijen and Whiteson [Seijen and Whiteson, 2009] showed that an agent can delay its update until the given state is revisited, however their method is too limited for our use.

4.2 The Delay function

A measure of the impact of observations needs to be developed to determine how many observations are necessary to change the suboptimal policy. The natural way of doing this is to find a function that maps (s_t, a_t, s_{t+1}) to a value, and then accumulate this value (set to be N in the algorithm in Sections 4.3) until it is large enough. Intuitively, at the start of the learning process the world to the agent is extremely unclear so every observation will have a large impact on the suboptimal policy. At the very end of the learning process the world to the agent is almost complete, so many observations are needed to change the suboptimal policy. What is needed is a function that is decreasing during the learning process to capture the value of observations. Dearden [Dearden *et al.*, 1998] has a way of computing value of information exploration over Q-Learning based on the theory of value of information. Since the observation counts for state-action pairs $\alpha_0(s_t, a_t)$ are non-decreasing during the learning process then $\frac{\delta}{(\alpha_0(s_t,a_t))^2}$, where δ is a constant, is a natural way of choosing that function. (Note that in this equation we used a square rather than the linear function. In Section 5 we show how this change will produce a theoretical time bound)

4.3 The Passive Dynamic Algorithm

The Passive Dynamic (PD) algorithm can be described as follows:

- Initialize B, $explore = 1$, $s = start$, $M = N = 0$
- **loop**
- Compute desirability by solving eq.'s 8 & 11
- Compute policy: if explore, use eq. 12, else eq. 10
- Select action, observe state
- if $delay = 0$, update B
- $N \leftarrow N + \frac{\delta}{\alpha_0(s_t,a_t)^2}$,
- if $N > enough \rightarrow delay = 0$ & $N = 0$
- $M \leftarrow M + (p(s'|s, b_{i-1}, a) - p(s'|s, b_i, a))^2$
- restart if $s = terminal$ unless $M > enough1$

5 Complexity Analysis

Now, let us consider the efficiency of the PD algorithm. Suppose there are a total of h steps in the whole learning process and n states of environment. It takes $O(n)$ time for iterating all of the states. Computing desirability takes $O(n^3)$ for solving the matrix equation. We show the total cost of the PD algorithm is $O(hn + n^4)$.

Theorem 1. *With delay, the total number of runs for updating the belief in the PD algorithm is $O(n)$.*

Proof. After h steps, the total accumulated N in the PD algorithm is at most $\delta(\frac{n}{1} + \frac{n}{2^2} + \frac{n}{3^2} + \cdots + \frac{n}{k^2})$, where $h = nk$. So the total number of updates is $\delta(\frac{n}{1} + \frac{n}{2^2} + \frac{n}{3^2} + \cdots + \frac{n}{k^2})/Const < \frac{n\delta}{Const}\sum_{i=1}^{\infty}\frac{1}{i^2} < \frac{2n\delta}{Const} = O(n)$ □

Since h is much bigger than n then $O(hn+n^4) \ll O(hn^3)$, where $O(hn^3)$ is the minimal complexity of Kolter's BEB Algorithm. Actually, when applying to the agent used in the experimental design, iterating states takes a constant amount of time so the complexity of the PD algorithm becomes $O(h+n^4)$, which means h and n were successfully separated. In other words, in the learning process time is a polynomial, since the cost h is just the time that the agent is interacting with the environment.

5.1 Pre-PD algorithm

The solution can be improved if there is additional information. If the transition probability (transition error) is the same in each state, that is, the transition error is due to the agent itself but independent of the environment then the transition probability can be excluded, and equation 9 becomes

$$uc(s'|s) = \frac{\sum_{a \in SS'} 1}{n_a} \tag{13}$$

where SS' is the set of actions that takes the agent from state s to s' ignoring the transition error. Now the passive dynamic is only related to the type of action and desirability is only related to the type of action and the reward function. Desirability in each state can be pre-computed. In the learning process only the transition matrix needs to be learned and projection

→	→	→	+2
↑	Obstacle	↑	-2
↑	←	↑	←
↑ (start)	↑	↑	←

Figure 2: Optimal policy in a 4×4 grid world

can be used to get an optimal action instead of having to solve any equations. By an argument similar to the above, we can show that the complexity of this approach is $O(n^3 + h + n)$, and the n^3 part can be done before the learning process.

6 Experimental Procedures

6.1 Grid World

Experimental Setting

Using the environment established in Section 1 set $m = 4$, 8, and 10. For the $m = 10$ case manually set 10 obstacles in the environment to be challenging. Beliefs were represented by Dirichlet count statistics. Each time the agent reached the terminal state was considered the end of a round and the agent was reset to the starting state.

We compared 4 algorithms in the experiment: 1) the BEB algorithm by Kolter, 2) the Delayed-BEB algorithm, which is the BEB algorithm plus a manual delay strategy, 3) the PD algorithm and 4) the Pre-PD algorithm.

Experimental Results

All four algorithms were tested to make sure that they converge to the optimal policy in the 4×4 case. For perfect belief, the optimal policy is illustrated in Figure 2. The four algorithms were then run using 100 rounds of exploration and the same optimal policy as shown above was returned and we tracked the increase in average reward per round during the learning process. Results in Figures 3 show that the PD and Pre-PD algorithms performed better than the others with smaller and nearly constant time cost for both 4×4 and 8×8 grids.

For the 10×10 grid world with 10 hard obstacles, performance improves for 20 rounds for three algorithms (Delayed BEB, PD and Pre-PD algorithms). However, the BEB algorithm could not arrive at an optimal solution within 100 rounds. The time costs of the Delayed BEB, PD and Pre-PD algorithms stayed nearly constant after 10 rounds rather than increasing linearly and the PD and Pre-PD algorithms performed the best. After 20 rounds the PD and Pre-PD algorithms performances were much more stable than the other two, because the algorithm had stopped the exploration process by the exploration switcher. In addition, as the complexity of the environment increased, the time cost for the PD and Pre-PD algorithms increased at a slower pace than the other algorithms, especially the Pre-PD algorithm, which increased linearly.

6.2 Cellular Telephone Systems

Cellular telephone is a more interesting problem that requires dynamically allocating communication resources (channels)

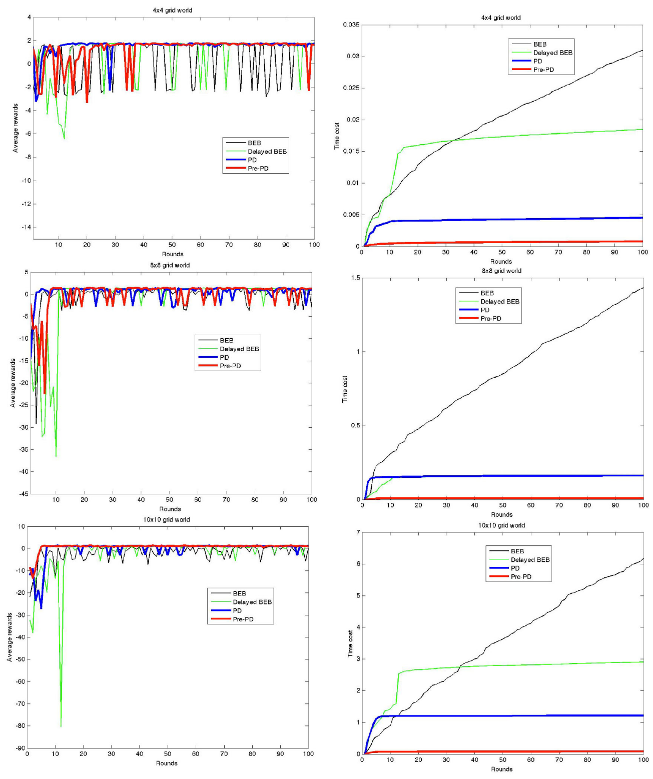

Figure 3: Average rewards & Time cost in 3 grid worlds for the four algorithms

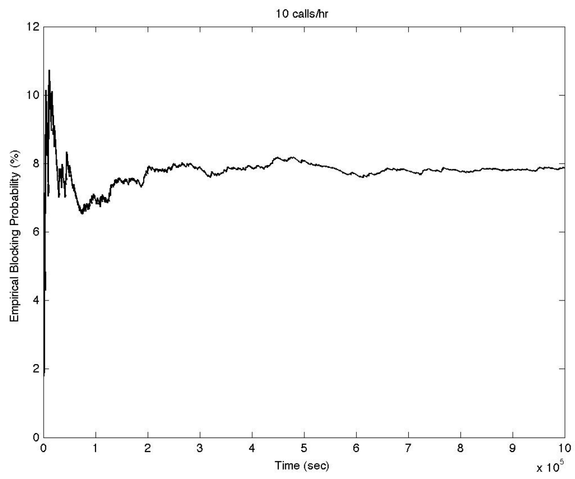

Figure 4: Blocking Probability of PD algorithm

so as to maximize service in a stochastic caller environment. Singh and Bertsekas [Singh and Bertsekas, 1997] presented a solution using Sutton's [Sutton, 1988] TD(0) with a feature-based linear network. Call arrivals are modeled as poisson processes with a separate mean for each cell, and call durations are modeled with an exponential distribution. We set the cellular array to 3 by 3 with 7 channels (roughly 7^9 configurations) and a channel reuse constraint of 3. We set the mean call arrival rates to 10 calls/hr respectively in each cell and the mean call duration is 3 minutes.

In order to use our PD algorithm, we need to establish the states, action, reward function, passive dynamics, exploration function and *belief* for the problem.

Since this problem has no terminal state (we set the state as Singh's way, and we will reduce the state to the small set later), we need to use the infinite horizon discounted cost case which makes Equation 8 becomes $z = QPz^{\gamma}$, where $Q = diag(\exp(-q)), P = UC$ and UC is the matrix of passive dynamics. It can be solved by using power iteration method [Todorov, 2009]. And here in this problem, we do not have transition error, and Action is just to choose the next available state given coming calls, so the equation 10 becomes $s' = \max_{s' \in S}\{z(s')\}$, where S is the set for next available states. The reward function of each state as the number of ongoing calls. The *belief* in this problem is just the parameters of all distributions (rate of coming calls & call duration). The passive dynamic is the natural change in calls without channel assignments, so that states with same ongoing calls on each cell are the same to our system. It means we can reduce the number of states from 7^9 to 16445.

The results of the PD algorithm are shown in Figure 4. We achieve a Blocking Probability of 8%, compared to the Singh and Bertsekas's RL method's 23% and Fixed Assignment method's 51% [Singh, 1997]. Note that about the first 1×10^5 seconds are the learning process, and it rapidly converges.

7 Conclusion

We provided a new efficient method for solving RL problems by exploiting linearly solvable MDPs and using limited policy updating and gave a polynomial time bound for the learning process. We also built a Pre-PD algorithm for a special class of RL problems in which the transition error is due to the agent itself rather than the environment, giving a linear learning time bound for the algorithm. We show our algorithm has excellent performance on both toy and test problems and that our algorithms' performances were more stable after convergence.

References

[Bellman, 2003] R. Bellman. *Dynamic programming*. Dover Publications, 2003.

[Dearden *et al.*, 1998] R. Dearden, N. Friedman, and S. Russell. Bayesian q-learning. In *National Conference on Artificial Intelligence*, pages 761–768. John Wiley & Sons Ltd, 1998.

[Kaelbling *et al.*, 1996] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.

[Kolter and Ng, 2009] J.Z. Kolter and A.Y. Ng. Near-bayesian exploration in polynomial time. In *International Conference on Machine Learning*, pages 513–520. ACM, 2009.

[Pitman and Yor, 1997] J. Pitman and M. Yor. The two-parameter poisson-dirichlet distribution derived from a stable subordinator. *The Annals of Probability*, 25(2):855–900, 1997.

[Poupart, 2007] P. Poupart. Tutorial on bayesian methods for reinforcement learning. In *ICML*, 2007.

[Puterman, 1994] M.L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc., 1994.

[Seijen and Whiteson, 2009] H.V. Seijen and S. Whiteson. Postponed updates for temporal-difference reinforcement learning. In *International Conference on Intelligent Systems Design and Applications*, pages 665–672. IEEE, 2009.

[Singh and Bertsekas, 1997] S. Singh and D. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. *Advances in neural information processing systems*, pages 974–980, 1997.

[Singh, 1997] S. Singh. Dynamic channel allocation in cellular telephones: a demo. http://web.eecs.umich.edu/~baveja/Demo.html, 1997.

[Strens, 2000] M.J.A. Strens. A bayesian framework for reinforcement learning. In *International Conference on Machine Learning*, pages 943–950, 2000.

[Sutton, 1988] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[Teh *et al.*, 2006] Y.W. Teh, M.I. Jordan, M.J. Beal, and D.M. Blei. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006.

[Thrun, 1992] S.B. Thrun. The role of exploration in learning control with neural networks. In *In Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, 1992.

[Todorov, 2009] E. Todorov. Efficient computation of optimal actions. *Proceedings of the national academy of sciences*, 106(28):11478–11483, 2009.