Efficient Learning in Linearly Solvable MDP Models

A THESIS

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

Ang Li

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Prof. Paul Schrater

June, 2012

© Copyright by Ang Li, 2012

All Rights Reserved

Acknowledgements

I wish to thank my committee members who were more than generous with their expertise and precious time. Thank you Prof. Paul Schrater, Prof. Maria Gini, and Prof. Wayne Richter for agreeing to serve on my committee.

I would like to acknowledgement and thank my school division for allowing me to conduct my research and providing any assistance requested. Special thanks goes to the members of staff of Computer Science department for their continued support.

Finally, I would like to thank the Director of my thesis, Prof. Paul Schrater, for his jovial encouragement and academic support.

Dedication

This thesis is dedicated to my family. A special feeling of gratitude to my loving parents, Xinjian Wang and Suhua Li whose words of encouragement and push for tenacity ring in my ears.

This thesis is also dedicated to many of my friends who have supported me throughout the process. I will always appreciate all they have done.

I dedicate this work and give special thanks to my girl friend Ruirui Mao for being there for me throughout the entire process of writing thesis.

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, inspired by Todorov's way of computing optimal action, we showed 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, the computational cost of learning was reduced using intermittent Bayesian updating reducing the frequency of solving the Bellman equation (either the normal form or Todorov's form). We also gave a polynomial theoretical time complexity bound for the convergence of the learning process of our new algorithm, and applied this directly to a linear time bound for the subclass of the reinforcement learning (RL) problem via MDP models with the property that the transition error depends only on the agent itself. Test results for our algorithm in a grid world were 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.

Table of Contents

List of Tables

List of Figures

I. Introduction

The topic of reinforcement learning (RL) has a long history in machine learning which in turn plays a crucial role in artificial intelligence. Researchers have tried many methods to model the learning problem, and the Markov Decision Process (MDP) model has proven to be the best one up to now. Based on the MDP model, researchers have developed several algorithms to solve learning problems such as value iteration [1, 2], policy iteration [1, 2], model-free algorithms (e.g. adaptive heuristic critic algorithms and Q-learning) [3] and model-based algorithms (e.g. the Dyna algorithm, the Queue-Dyna algorithm and the RTDP algorithm) [3]. The model-based algorithm is more popular because it has a good structure for the learning process and a good exploration process. However, the high computational cost of solving the Bellman equation [2] in model-based algorithms limits the application of RL.

The first limitation is the cost of solving the non-linear Bellman equation per iteration, which will be even higher if the belief state is considered. Recently, Todorov proposed a way of simplifying the Bellman equation into a linear equation [4]. He showed that the Bellman equation for the subclass of control problems with a certain structure in the problem formulation can be greatly simplified, and that most discrete control problems have this structure. Most importantly, he gave a way of calculating the desirability (D) of each state by using only the passive dynamic and the reward function. Therefore, if the passive dynamic and the reward function are unchanged along with the environment then the linear Bellman equation needs to be pre-solved for only one iteration. However, this is only a way of choosing an optimal action without learning. In this paper, we

focused on combining the learning process with Todorov's result, and added an exploration strategy to trade-off exploration and exploitation.

The second element that increases cost is the effect of facts on optimal policy. A set of Bellman equations needs to be resolved, finding the optimal policy for the given observation for several iterations and, since every iteration new facts are observed, the facts need to be updated immediately. However, facts are observed each time but the optimal policy may not change very much. Therefore, resolving the Bellman equation each iteration is inefficient, so delaying the updating of the observation may be a way to reduce the computation complexity. However, the convergence speed of the policy is highly related to updating observations, so how to delay it and how long it can be delayed become crucial. In our research we found a function to control the updating process and gave a time bound for solving the Bellman equation using this function.

II. Problem Description

2.1 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, $R:S\times A\to\Re$ 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_t is the reward gathered at time t and γ is a discount factor. This problem can be solved by solving a set of Bellman equations [2]

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

and selecting the action inside the max operator as optimal policy. We can apply classical algorithms such as Value Iteration or Policy Iteration to find a solution to this equation [5].

2.2 Reinforcement Learning via MDP Models

In RL, the above model, in which neither the transition function nor the reward function is known, is used. For this research the reward function was assumed known with only

T needing to be estimated. In other words, the MDP model computes an optimal policy, and the RL process learns the optimal policy.

2.3 Example of a Learning Problem

Suppose that an agent is situated in an m × m (e.g. m=4, 8, 10) grid world (the 4x4 case 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 called 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, have obstacles associated with them that have a probability of 0.1 (p=0.1).

			+2
	Obstacle		-2
Start			

Figure 1: A 4x4 grid world

The environment has a transition error: the correct outcome occurs with p=0.8, and the agent moves at right angles to the correct direction with p=0.2. (e.g., if we choose the action up, we have p=0.8 to go up but p=0.1 to go either left or right). Moreover, 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. An RL round creates a moving set from the start state to the terminal state.

III. Related Work

3.1 Bayesian Reinforcement Learning

Strens proposed a Bayesian solution for RL using the MDP model [6], which is to explicitly maintain a *belief* over models. *Belief* is denoted by B, where $B = P(T)$. After making an observation, B is updated using Bayes' rule, $B_{t+1}\left|s_t, a_t, s_{t+1} \propto p(s_t, a_t, s_{t+1}|T) \times B_t\right.$. In this algorithm B about the transition distributions is maintained by a set of Dirichlet distributions over each T. With the addition of B and defining the state by the tuple (b,s), the Bellman equation becomes

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

(2)

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.

3.2 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 proposed an approximation algorithm called the Bayesian Exploration Bonus Algorithm (BEB) [7] 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

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

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.

3.3 Efficient Computation of Optimal Actions

The problem with equation 3 is that the model-based computation of optimal actions is usually intractable and typically requires extensive approximation. However, efficient and exact computation of optimal actions is possible. Recently, Todorov proposed a way of simplifying the Bellman equation into a linear equation for a subclass of MDP models with a certain structure on the problem formulation [4]. 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

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

where $E[v(s')]$ is the expected cost-to-go at the next state. Todorov allowed the agent to specify the transition probabilities $u(s'|s)$ directly instead of asking the agent to specify symbolic actions, which are then replaced with the transition probabilities. Now

$u(s'|s) = p(s'|s,a)$ and the agent has the power to reshape the dynamics in any way it wishes. Let $uc(s'|s)$ denote the passive dynamics characterizing the behavior of the system in the absence of controls. The action 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 $l(s,a)$ has the form

$$l(s,a) = q(s) + KL(u(\cdot|s) \| uc(\cdot|s)) = q(s) + E_{s'\sim u(\cdot|s)}[\log \frac{u(s'|s)}{uc(s'|s)}] \tag{5}$$

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

$$-\log(z(s)) = q(s) + \min_a \{E_{s'\sim u(\cdot|s)}[\log \frac{u(s'|s)}{uc(s'|s)z(s')}]\} \tag{6}$$

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,

$$z(s) = \exp(-q(x))g[z](s) \tag{7}$$

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 (7) 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 said this is a more significant limitation, but for the 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 while ensuring that exp(-q(x)) does not become numerically zero.

3.4 Postponed Updates for Temporal-Difference RL

Seijen and Whiteson demonstrated the effects a manipulative delay component had on the Temporal Difference (TD) method by recording the agent's last-visit experience [8].

The agent can delay its updating process until the given state of observation is revisited. There are some disadvantages to this modification that will be discussed in the next section.

IV. Motivation

In order to adequately explore the environment, ideally we would execute a full Bayesian look-ahead since the Bayesian solution (see section 3.1) automatically trades-off exploration and exploitation. Exploitation maximizes rewards using current estimates and exploration maximizes long-term rewards and they are both an indispensable part of the RL process. However, the Bayesian solution is intractable in general (see section 4.1). Moreover, Todorov's approach to solving MDPs only works with known transition probabilities. To add learning to this approach, the Bayesian look-ahead needs to be approximated. Several approximation methods have been proposed, including Dearden, Strens, etc [9].

4.1. Intractability

The perfect solution to RL with MDP models is intractable for two reasons. First, solving 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 it will affect the next *belief* state. Second, equation (2) needs to be solved several times, since after each iteration we get an observation by interacting with the environment and the *belief* state needs to be updated, resolving equation (2) to get the optimal policy. What is needed is to speed up the time of solving the Bellman equation to get the suboptimal action more efficiently and to reduce the number of iterations required for resolving the Bellman equation.

4.2. Value Iteration (Policy Iteration) vs. Matrix Multiplication

The BEB algorithm presented by Kolter and Ng handles the problem of the first issue from Section 4.1 very well (see section 3.2) [7]. However, equation (3) is still not linear, and must be solved by value iteration. Therefore it is still not sufficient enough and we cannot get a good time complexity bound for value iteration. Fortunately, Todorov provided a linear solution to the Bellman equation that is suitable for most discrete control problems. More importantly, the solution gives a method of calculating the desirability of each state using only the passive dynamic and the reward function. Therefore, if the passive dynamic and the reward function are unchanged with the environment all that is needed is to pre-solve a linear Bellman equation once. However, the result does not contain any learning process, it is a pure solution to the Bellman equation. One of the purposes of this research was to combine the learning process with Todorov's result. For the traditional Bayesian solution to the RL problem (see section 3.1), the learning process will automatically trade-off exploration and exploitation based on the structure of equation (2). Unfortunately, this balance will be broken due to the new formulation, so a function is needed to balance exploration and exploitation. Moreover, the formulation of Todorov's result allows the passive dynamic into the system, for some of the problem, and we can pre-calculate the passive dynamic in order to give the agent more information about the environment.

4.3. Time Cost ($O(h \times n^3)$) and the Manual Delay of Updating

The next task is to combine Todorov's result with the RL process. If there are n states total and h exploration steps are needed for converging, then at each step equation (7)

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 be large, and then the algorithm is still inefficient. What is wanted is to separate h and n in time complexity as in the form $O(h + n^3)$. Notice that in each iteration a new observation does not necessarily change the suboptimal action so resolving equation (7) every time is inefficient. What we want is to find a control function that controls the updating process such that the total number of updates (i.e. resolve equation (7)) is a function of n, so h and n can be separated. Seijen and Whiteson demonstrated the effects a manipulative delay component had on the TD method by recording the agent's last-visit experience [8]. The agent can delay its update until the given state is revisited, but there are some limitations to its use. First, it can only be implemented in the TD method, a model free method which makes extremely inefficient use of the data gathered [3]. Second, the method still does not have a good time bound for converging. Third, even if some observations are crucial, the method will still delay the update of the observation.

V. Passive Dynamics

A 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 dynamic, producing the equation

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

where n_a is the number of actions.

VI. Reinforcement Learning via a New Model

Now we have an efficient model to determine the optimal action using transition probability (Section 3.3). However, there is no learning part in it. In this section a method to determine the RL process via this new model will be discussed.

6.1 Choosing the Suboptimal Action from Desirability

The first issue of the learning process is how to choose an action from desirability given the current *belief* state, and using a projection function to choose it was deemed the best method. This produces the equation

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

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

6.2 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, $B_{t+1} \left| s_t, a_t, s_{t+1} \right. \propto p\left(s_t, a_t, s_{t+1} \middle| T \right) \times B_t$. 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

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

6.3 Exploration vs. Exploitation

The traditional Bayesian solution for an RL problem will automatically trade-off exploration and exploitation based on the structure of equation (2). Unfortunately, the balance is broken due to the new formulation, so a function is needed to balance exploration and exploitation. Similar to Kolter's method in Section 3.2, an exploration bonus was added when choosing the suboptimal action from equation (10), so equation (10) becomes

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

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 [10]. In order to trade-off exploration and exploitation a reasonable scheme for trade-off was developed, referred to as greedy in the limit of infinite exploration (GLIE). A smarter GLIE gives higher weights to actions not tried very often and gives lower weights to actions with little benefit. Thus equation (1) becomes

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

where $f(u,n)$ is the exploration function which increases with expected utility u and decreases with the number of tries n. The reason why Kolter's way of choosing the exploration function was used is the exploration bonus function is the simplest function that satisfies the conditions of the exploration function, and Kolter proved the

performance of this exploration bonus function. 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 7.2 & 8.2). 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, 8) Choosing the suboptimal action from equation (10) or (12) depending on the exploration switcher and 5) 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.

VII. Manual Delay of the Algorithm

7.1 The Delay function

Although the new algorithm and the BEB algorithm are acceptable when solving an RL problem within a simple environment, the computational cost during the learning process is still large in a complex environment. For example, in a 10×10 grid-exploration problem with hard obstacles the BEB needs more than one minute to get the optimal policy. The long processing time is a result of returning a new suboptimal policy caused by the need to solve equation (2) when an observation of the 3-tuple (s_t, a_t, s_{t+1}) has been updated. Some of the observations will change the suboptimal policy so little that solving the corresponding equations provides no benefit. A standard for 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 7.2 & 8.2) until it is large enough. Intuitively thinking, at the very beginning 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 has a way of computing value of information exploration over Q-Learning based on the theory of value of information [11]. 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 used in our previous work. In Section **VIII** we show how this change will produce a theoretical time bound)

7.2 The Passive Dynamic Algorithm

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

1. Set the initial *B*, set the total round R;

2. Set N = 0, M = 0, Enough = Constant, Enough1 = Constant,

   Exploration Switcher = True, Current state = Start state;

3. Obtain the suboptimal policy of the current state from equation (12) if Exploration Switcher =True, by equation (10) otherwise;

4. Choose the action based on current policy and current state, providing an observation of 3-tuple (s_t,a_t,s_{t+1}), updating *B*;

5. $N = N + \frac{\delta}{(\alpha_0(s_t,a_t))^2}, M = M + (p(s'|s,b_{i-1},a) - p(s'|s,b_i,a))^2$;

6. Current state = s_{t+1}, If the current state is the terminal state and M < Enough1, set Exploration Switcher = False, set Current state = Start state, else If Current state is terminal state, set Current state = Start state, else set M = 0; If the current state is the terminal state and the Current round = R, then halt;

7. If N > Enough then set N = 0, go to 8;

   else go to 3;

8. Updating the desirability by solving equations (11) and (8), then go to 3.

VIII. Complexity Analysis

8.1 Complexity of the PD algorithm

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. Step 3 takes $O(n)$ time for iterating all of the states and in the environment of Section **IX** this cost is constant since there are only 4 potential states for iterating. Steps 4 through 7 take a constant amount of time. Step 8 takes $O(n^3)$ for solving the matrix equation. Therefore the total cost of the PD algorithm is $O(hn+n^4)$ given Theorem 8. Since h is much bigger than n then $O(hn+n^4) >> O(hn^3)$, which is the complexity of Kolter's BEB Algorithm. Actually, when applying to the agent used in the experimental design, the step 3 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.

Theorem 8 The total number of runs for step 8 in the PD algorithm is $O(n)$.

Proof: After h steps, the total accumulated N in the PD algorithm is at most $\frac{n}{1}+\frac{n}{2^2}+\frac{n}{3^2}+\cdots+\frac{n}{k^2}$, where $h=nk$. So the total number of runs for step 8 is

$$
\begin{aligned}
&(\frac{n}{1}+\frac{n}{2^2}+\frac{n}{3^2}+\cdots+\frac{n}{k^2})/Const \\
&<\frac{n}{Const}\sum_{i=1}^{\infty}\frac{1}{i^2} \\
&<\frac{2n}{Const} \\
&=O(n)
\end{aligned}
$$

8.2 Pre-PD algorithm

The solution can be improved if there is additional information. The optimal solution is only related to the passive dynamic and the reward function and the passive dynamic has the form of equation (9). If the transition probability (transition error) is the same in each state (the exact value is not necessary, just knowing they are the same is enough), that is, the transition error is due to the agent itself but independent of the environment (in this environment that is the case since, no matter in which state, the agent has the same transition probability), 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 can be used to get an optimal action instead of having to solve any equations.

The Pre-computed Passive Dynamic algorithm is described as follows:

0. Pre-Calculate desirability using equations (13) and (8);

1. Set the initial *Belief*, set the total number of rounds to R;

2. Set N = 0, M = 0, Enough = Constant, Enough1 = Constant,

Exploration Switcher = True, Current state = Start state;

3. Get the suboptimal policy of the current state using equation (12) in this case as Exploration Switcher =True, use equation (10) otherwise;

4. Choose the action based on Current policy and Current state, getting an observation of the 3-tuple (s_t, a_t, s_{t+1});

5. $N = N + \frac{\delta}{(\alpha_0(s_t, a_t))^2}, M = M + (p(s' \mid s, b_{i-1}, a) - p(s' \mid s, b_i, a))^2$;

6. Current state = s_{t+1}, If Current state is the terminal state and M < Enough1, set Exploration Switcher = False, set Current state = Start state; else If Current state is the terminal state, set Current state = Start state, else set M = 0; If Current state is the terminal state and Current round = R, then halt;

7. If N > Enough, then set N = 0, go to 8;

else go to 3;

8. Update *Belief*, then go to 3.

8.3 Complexity of the Pre-PD Algorithm

Similar to the PD algorithm, the efficiency of the Pre-PD algorithm needs to be considered. Suppose there are h steps in the whole learning process and n states of

environment. Step 0 (pre-calculation) takes $O(n^3)$ time. Step 3 takes $O(n)$ time for iterating all of the states and in the environment of Section **IX** this cost is constant since there are only 4 potential states for iterating. Steps 4 through 7 take a constant amount of time. Step 8 takes only a constant amount of time for updating (see section XII to understand why updating *belief* was moved into the delay part). Therefore the total cost of the Pre-PD algorithm is $O(n^3 + hn + n)$, given Theorem 8. Since h is much bigger than n then $O(n^3 + hn + n) >> O(hn^3)$, which is the complexity of Kolter's BEB Algorithm. Like the agent we will use in our experiment, step 3 takes only a constant amount of time, so the complexity of the Pre-PD algorithm is simply $O(n^3 + h + n)$, and that means we successfully separated *h* and *n*. Most importantly, the learning process includes only a linear function for time.

IX. Experimental Procedures

9.1 Experimental Setting

Using the environment established in Section 2.3 set m = 4, 8 and 10. For the m = 10 case manually set 10 obstacles in the environment to be hard.

9.2 Application to the Algorithms

The state, reward function and action are the same as discussed in section 2.3. Since the concern was with the setting of discrete states and actions, a natural way of representing *belief* is to let b consist of a set of Dirichlet distributions. In order to simplify the experimental process, we used the observations count rather than sampling from a Dirichlet distribution of observations. This results in

$$b = \{\alpha(s,a,s')\},\ P(s'|b,s,a) = \frac{\alpha(s,a,s')}{\alpha_0(s,a)},$$

where $\alpha_0(s,a) = \sum_{s'} \alpha(s,a,s')$ and $\alpha(s,a,s')$ is the number of observations of 3-tuple (s,a,s').

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.

The values of the constants ($\delta, \beta, enough, enough1$) for each experiment and for each algorithm are listed in the Appendix.

X. Experimental Results

All four algorithms were tested to make sure that they converge to the optimal policy in the 4x4 case. If the agent is assumed to know the transition error then the MDP models can be used without *belief* to find a solution, and the optimal policy produced is that illustrated in Figure 2.

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

Figure 2: Optimal policy in a 4x4 grid world

The four algorithms were then run using 100 rounds of exploration and the same optimal policy as shown above was returned.

A slight change in optimal policy will have little effect on the reward per round. So the concern was more about the increase of the average reward per round during the learning process. For each case, m=4, 8 and 10, a comparison between the effects the four algorithms had on average reward, figures 3, 5 and 7, and time cost per round, figures 4, 6 and 8, are illustrated.

In a 4x4 grid world, figures 3 & 4, after about 30 rounds the four algorithms started to perform well. But the time costs of the Delayed BEB, PD and Pre-PD algorithms stayed nearly constant after 20 rounds rather than increasing linearly, and the PD and Pre-PD algorithms performed better than the others. In this simple case both algorithms were efficient so the advantages of the PD and Pre-PD algorithms were not obvious. After 40

rounds the PD and Pre-PD algorithms performances became much more stable than the other two as the algorithm had stopped the exploration process by the exploration switcher.

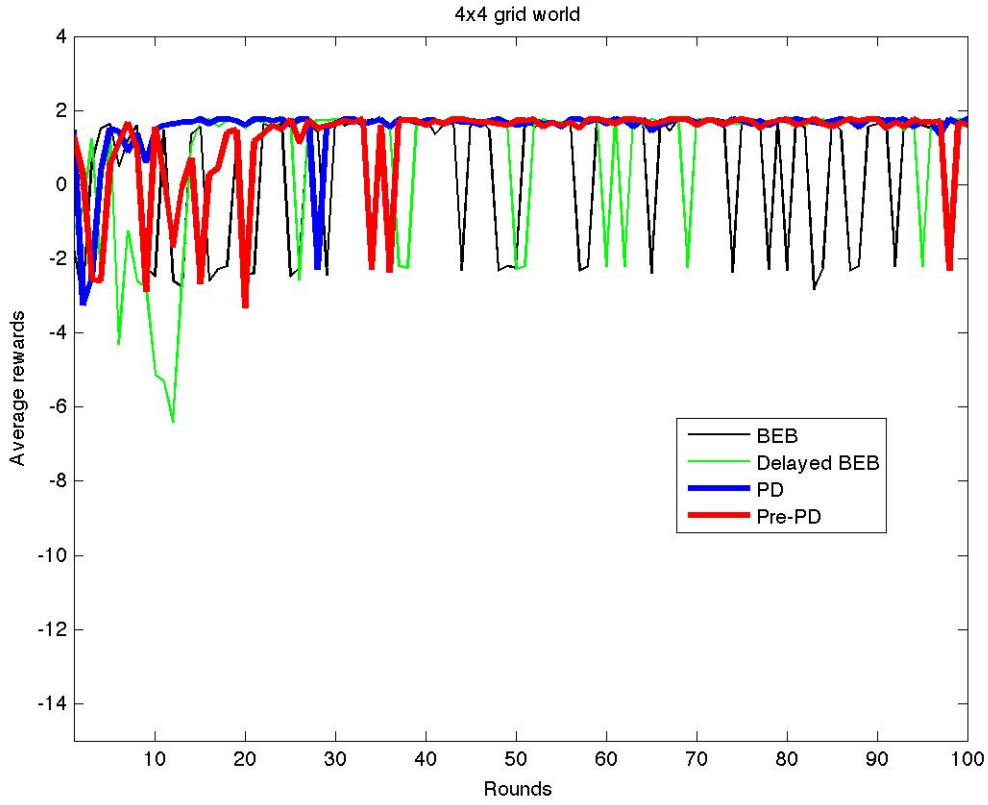

Figure 3: Average rewards in a 4x4 grid world for the four algorithms

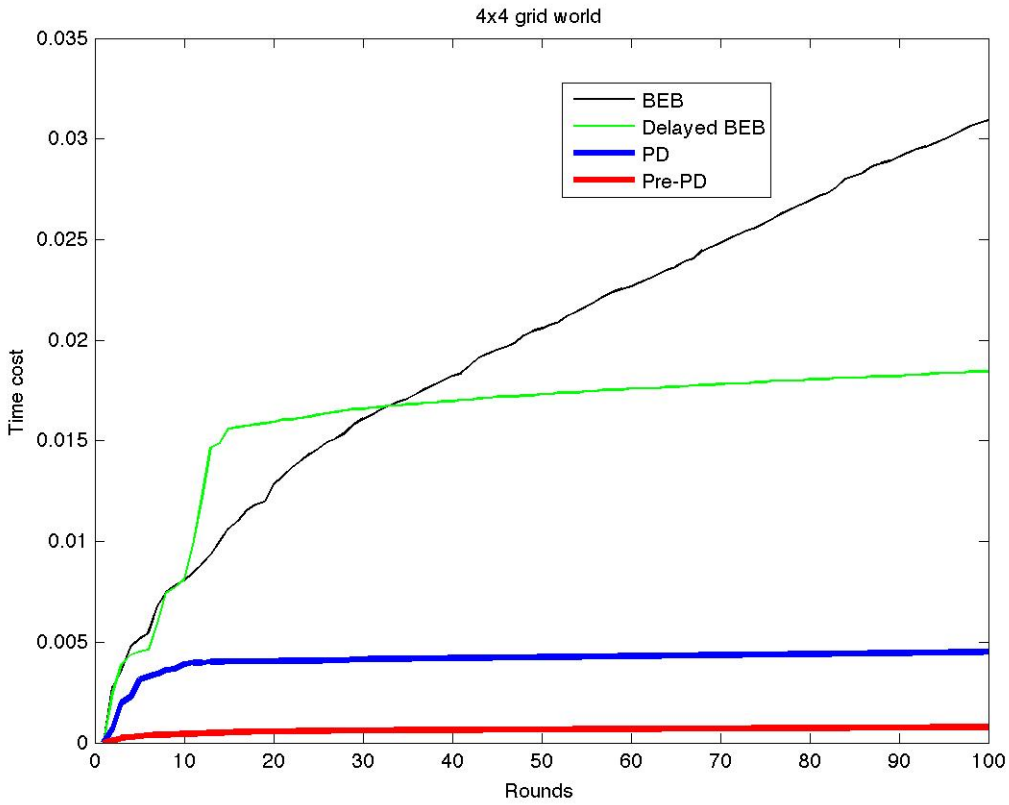

Figure 4: Time cost in a 4x4 grid world for the four algorithms

The results from an 8x8 grid world are shown in figures 5 and 6. After about 20 rounds the four algorithms started to perform well. The time costs of the Delayed BEB, PD and Pre-PD algorithms stayed nearly constant after 10 rounds rather than increasing linearly; the Pre-PD algorithm had the best performance. The stability of the performance after convergence was the same as only one obstacle was placed into this large environment so there was a greater chance of getting a transition error.

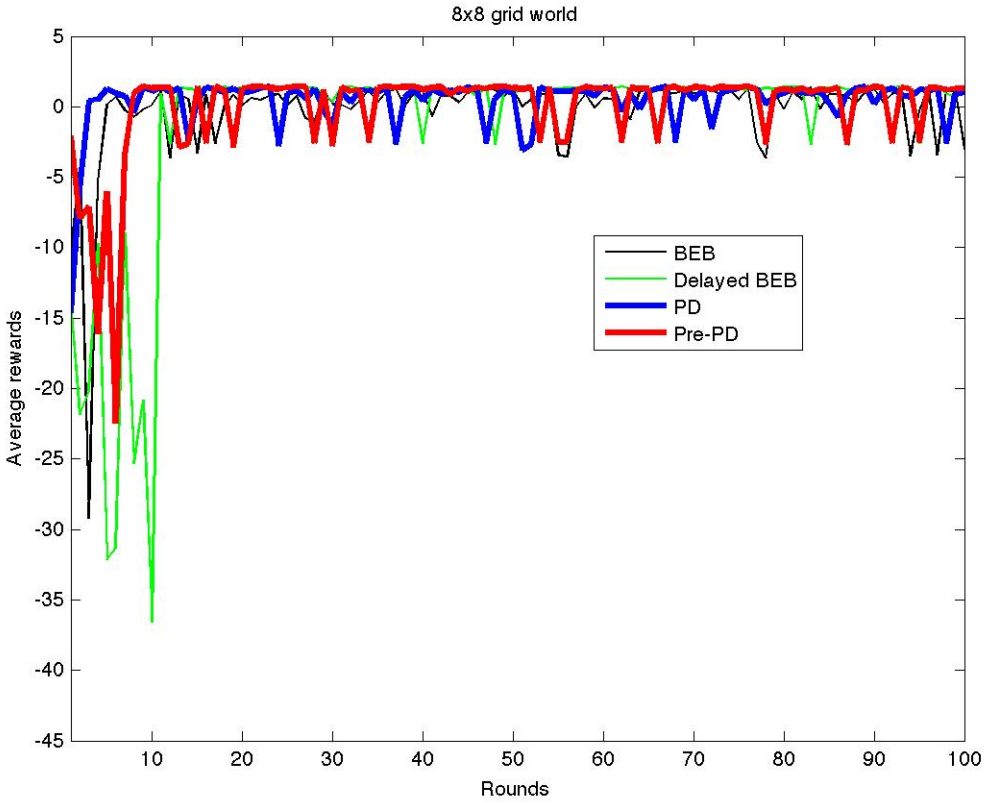

Figure 5: Average rewards in an 8x8 grid world for the four algorithms

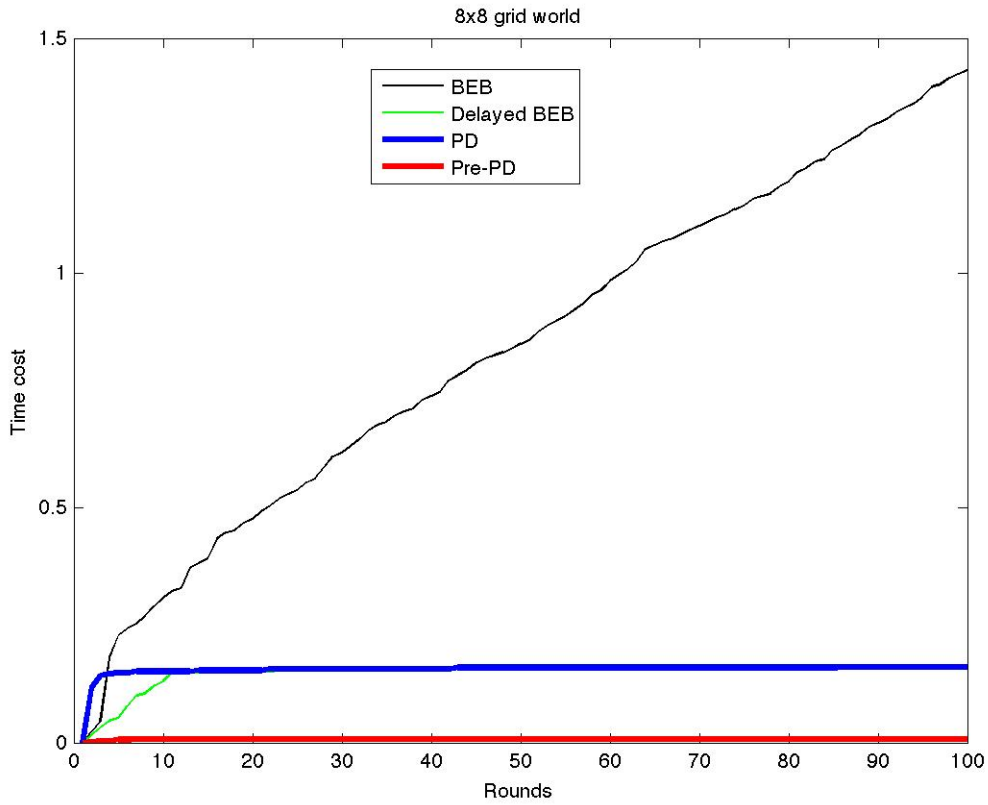

Figure 6: Time cost in an 8x8 grid world for the four algorithms

The final run was a 10x10 grid world with 10 hard obstacles and the results are illustrated in figures 7 and 8. After about 20 rounds three algorithms (Delayed BEB, PD and Pre-PD algorithms) started to perform well. This the most difficult case and 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, again 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.

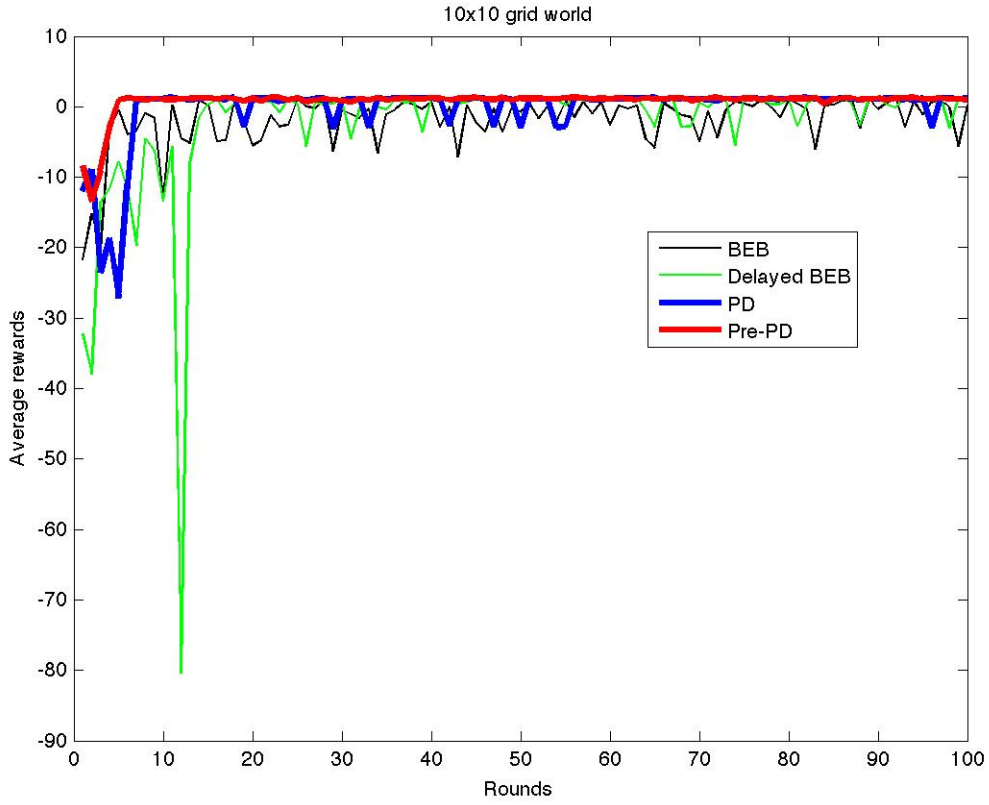

Figure 7: Average rewards in a 10x10 grid world for the four algorithms

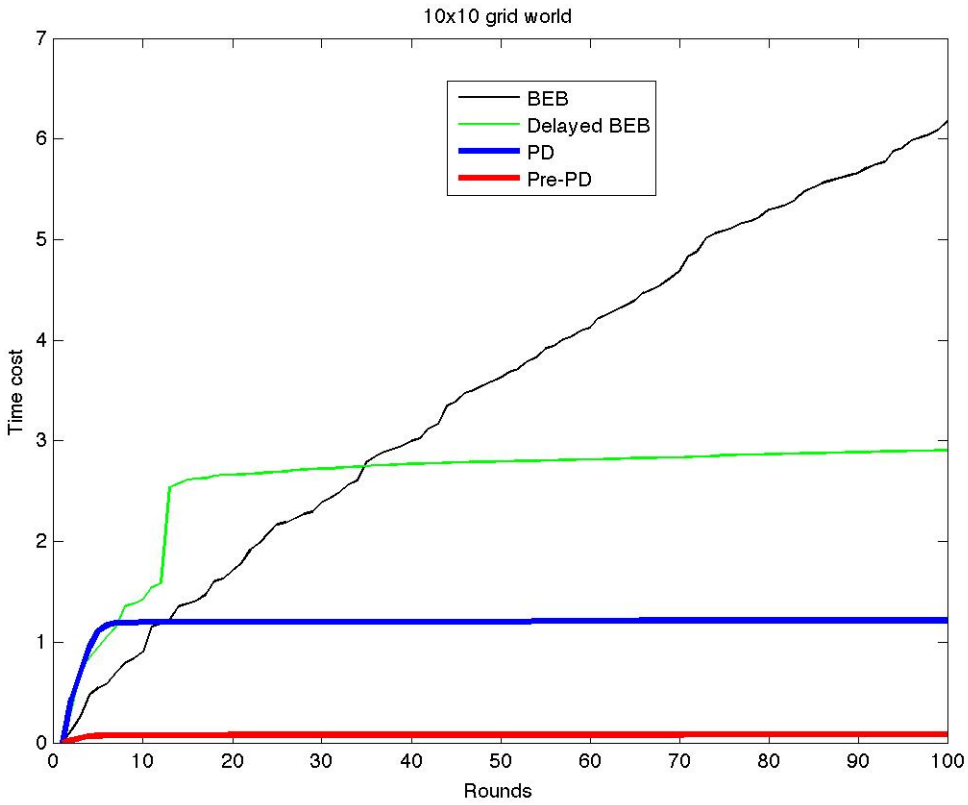

Figure 8: Time cost in a 10x10 grid world for the four algorithms

XI. Conclusion

In this paper we pointed out two reasons why the perfect solution for RL using MDP models is intractable. First, solving equation (1) is extremely hard, since in the equation the optimal action is based not only on how it will affect the next state but also on how it will affect the next *belief* state. Second, equation (1) must be solved several times, since during each iteration an agent interacts with the environment and *belief* must be updated, resolving equation (1) to get the suboptimal policy. Inspired by Todorov's efficient computation of an optimal action we proposed a PD algorithm. In the PD algorithm, we constructed a passive dynamic for the MDP model and combined the learning process with Todorov's solution to get an efficient way of solving RL problems. We also used a decreasing function to capture the property that observations at the beginning of learning were valuable while those at the end of learning were not, so that the updating process could be manually delayed. Most importantly, we 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 tried our two algorithms on a grid world comparing them with the BEB and Delayed-BEB algorithms and the experimental results showed that our algorithms were more efficient than the BEB algorithm without losing convergence speed. And as the environment got more and more complex, the advantage of the PD and Pre-PD

algorithm became clearer. We also show that our algorithms' performances were more stable after convergence.

XII. Future Work

There are several open research questions regarding the PD algorithm. First, Todorov's solution was that policy separates into a goal term and a dynamics term, which means we can place separate priors on the passive dynamic and the goal that act as structured priorities on policies . Second, the passive dynamic allows us to give more pre-information to the agent, which can accelerate the learning speed (the Pre-PD Algorithm is an example). Therefore, how to construct the passive dynamic and what information it contains becomes crucial. Third, in the algorithm we have four constants to be determined that are crucial for learning speed and performance and further research is required to efficiently determine the best parameters.

In the Pre-PD algorithm we moved updating *belief* into the delay updating part, since in our experiment we used a simple estimation of *belief* instead of maintaining a set of Dirichlet distributions. Thus updating *belief* cannot be done in constant time so the algorithms need to be tested with *belief* as a Dirichlet distribution [12, 13].

In addition, research on the manual updating of observations is rare as researchers tended to focus more on how to solve the Bellman equation efficiently or how to build an approximate Bayesian model. However, we still have at least two potential solutions for this problem. First is to find more precise functions to capture the value of the observation. Second is the methods used in the delayed environment in which the reward function or action may be delayed for some reason. In such an environment, researchers have found some particular algorithms to solve it [14, 15]. Based on this, we can try to find the relation between manual delay and forced delay.

Reference

[1] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*: Prentice Hall, 2010.

[2] R. Bellman, *Dynamic Programming*: Dover Publications, 2003.

[3] L. P. Kaelbling*, et al.*, "Reinforcement Learning: A Survey," *Journal of Aritificial Intelligence Research,* vol. 4, 1996.

[4] E. Todorov, "Efficient computation of optimal actions," *Proceedings of the National Academy of Sciences,* vol. 106, 2009.

[5] M. L. Putterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 2005.

[6] M. J. A. Strens, "A Bayesian Framework for Reinforcement Learning," in *International Conference on Machine Learning*, 2000.

[7] J. Z. Kolter and A. Y. Ng, "Near-Bayesian exploration in polynomial time," in *International Conference on Machine Learning*, 2009.

[8] H. V. Seijen and S. Whiteson, "Postponed Updates for Temporal-Difference Reinforcement Learning," in *International Conference on Intelligent Systems Design and Applications*, 2009.

[9] P. Poupart, "Tutorial on Bayesian Methods for Reinforcement Learning," in *ICML*, 2007.

[10] 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.

[11] R. Dearden*, et al.*, "Bayesian Q-Learning," in *Fifteenth National Conf. on Artificial Intelligence (AAAI)*, 1998.

[12] Y. W. Teh*, et al.*, "Hierarchical Dirichlet Processes," *JASA,* vol. 101, 2006.

[13] J. Pitman and M. Yor, "The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator," *Annals of probability,* vol. 25, 1997.

[14] T. J. Walsh*, et al.*, "Planning and Learning in Environments with Delayed Feedback," in *Machine Learning*. vol. 4701, 2007.

[15] E. Schuitema*, et al.*, "Control delay in Reinforcement Learning for real-time dynamic systems: A memoryless approach," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

Appendix

Value of Constants used in experiment design

	δ	β	*enough*	*enough1*
BEB	5	N/A	N/A	N/A
Delayed BEB	5	10	5	N/A
PD	5	10	5	1
Pre-PD	5	10	5	1

Table 1: Value of Constants used in experiment design for the 4x4 case

	δ	β	*enough*	*enough1*
BEB	1	N/A	N/A	N/A
Delayed BEB	1	10	20	N/A
PD	40	10	40	60
Pre-PD	30	10	20	60

Table 2: Value of Constants used in experiment design for the 8x8 case

	δ	β	*enough*	*enough1*
BEB	1	N/A	N/A	N/A
Delayed BEB	10	10	10	N/A
PD	40	10	40	60
Pre-PD	30	10	1	100

Table 3: Value of Constants used in experiment design for the 10x10 case