PREDICTION FOR CONTROL DELAY ON REINFORCEMENT LEARNING

Junya Saito, Kazuyuki Narisawa and Ayumi Shinohara
Graduate School of Information Sciences, Tohoku University, Sendai-shi, Japan

Keywords: Machine learning, Reinforcement learning, Control delay, Markov decision process.

Abstract: This paper addresses reinforcement learning problems with constant control delay, both for known case and unknown case. First, we propose an algorithm for known delay, which is a simple extension of the model-free learning algorithm introduced by (Schuitema et al., 2010). We extend it to predict current states explicitly, and empirically show that it is more efficient than existing algorithms. Next, we consider the case that the delay is unknown but its maximum value is bounded. We propose an algorithm using accuracy of prediction of states for this case. We show that the algorithm performs as efficient as the one which knows the real delay.

1 INTRODUCTION

Reinforcement learning is one of the most active research area of machine learning, where an agent learns how to take appropriate actions in an environment so that it obtains the maximum cumulative reward through observation and interaction. In many cases, the Markov Decision Process (shortly MDP) is used as a framework for reinforcement learning.

In real world applications, it is often the case that time delay, called *control delay* between observation and control matters seriously. For example, communication latency between controlling program and target robots may not be negligible, especially for slow networks, such as Mars exploration project. Even for a single agent, decision making including heavy computations, e.g. image recognition, may cause action delays.

Some researchers have investigated such situations, that is *reinforcement learning with delay*. Katsikopilos *et al.* showed that the MDP with delay can be reduced to the MDP without delay (Katsikopoulos and Engelbrecht, 2003). Moreover, they showed that delay for actions and delay for observations are equivalent from a view point of learner, although these have been developed separately. Walsh *et al.* proposed Model Based Simulation method, in which they combined model-based reinforcement learning and prediction of current states. Their algorithm performs well even under the delay, although it requires heavy computational resources (Walsh et al., 2007). Schuitema *et al.* approached this problem by using model-free learning (Schuitema et al., 2010). Their algorithm, called dSARSA(λ), is based on Sarsa(λ) (Sutton and Barto, 1998), and it updates Q-function with considering the delay. However, it did not explicitly use the delay for selecting next actions. In this paper, we show that the performance will increase if we add prediction of states in dSARSA(λ) by some experiments.

Moreover, we also consider the case that the delay is not known to the learner. We propose a simple algorithm using accuracy of prediction of states, and we verify that it works as efficient as the learner who knows the real delay.

2 PRELIMINARIES

A *Markov Decision Process* (MDP) is a 4-tuple $\langle S, A, P, R \rangle$, where S is a set of *states*, and A is a set of *actions*. $P\colon S \times A \times S \to [0,1]$ is a mapping indicating the probability that the next state becomes $s' \in S$ when an agent executes action $a \in A$ in state $s \in S$. The *reward function* $R\colon S \times A \to \Re$ defines the expected reward that the agent obtains when taking action $a \in A$ in state $s \in S$. We assume that both S and A are finite and discrete.

In this paper, we deal with control delay (Schuitema et al., 2010), which is delay between observation and control action. We refer to control delay as delay below. Walsh et al. (Walsh et al., 2007) proposed the *constant delayed MDP* (CDMDP),

Saito J., Narisawa K. and Shinohara A..
PREDICTION FOR CONTROL DELAY ON REINFORCEMENT LEARNING.
DOI: 10.5220/0003883405790586
In *Proceedings of the 4th International Conference on Agents and Artificial Intelligence* (SSML-2012), pages 579-586
ISBN: 978-989-8425-95-9

which is an MDP with known constant delay. We describe CDMDP as a 5-tuple $\langle S, A, P, R, k \rangle$, where k is a non-negative integer representing *delay*. We are also interested in the situation that the delay is *unknown* to the agents, although its maximum value is known. We define an *unknown constant delayed MDP* (UCDMDP) as a 5-tuple $\langle S, A, P, R, k_{\max} \rangle$, where $k_{\max}$ is a non-negative integer that bounds delay. The real value k of delay is not given to the agent, but is fixed and satisfies $0 \leq k \leq k_{\max}$.

3 dSARSA$(\lambda)_k$: ALGORITHM FOR KNOWN DELAY

Q-learning and Sarsa are popular on-line algorithms which directly estimate the Q-function $Q(s,a)$, that calculates the quality of a state-action combination. In order to accelerate convergence, *eligibility traces* are often combined to Q-learning and Sarsa, (see, e.g., (Sutton and Barto, 1998)) that is called $Q(\lambda)$ and Sarsa(λ), respectively.

Several approaches using $Q(\lambda)$ and Sarsa(λ) are possible to tackle with the delay k. Due to (Katsikopoulos and Engelbrecht, 2003), if the state space of the MDP is expanded with the actions taken in the past k steps, a CDMDP is reducible to the regular MDP $\langle S \times A^k, A, P, R \rangle$. It implies that normal reinforcement learning techniques are applicable, for small k. However, if k is large, the state space grows exponentially, so that the learning time and memory requirements would be impractical. If we treat $\langle S \times A^k, A, P, R \rangle$ as if $\langle S, A, P, R \rangle$, the problem belongs to the Partially Observable MDPs (POMDPs). In (Loch and Singh, 1998), they showed that Sarsa(λ) performs very well for POMDPs.

In (Schuitema et al., 2010), they refined the update rule of $Q(s,a)$ by taking the delay k into the consideration explicitly;

$$Q(s_n, a_{n-k}) \quad \leftarrow \quad Q(s_n, a_{n-k}) + \alpha \cdot \delta_n,$$

where α is the learning rate,

$$\delta_n = \begin{cases} r_{n+1} + \gamma \cdot \max_{a' \in A} Q(s_{n+1}, a') - Q(s_n, a_{n-k}) \\ \qquad\qquad\qquad\qquad \text{for } Q\text{-learning} \\ r_{n+1} + \gamma \cdot Q(s_{n+1}, a_{n-k+1}) - Q(s_n, a_{n-k}) \\ \qquad\qquad\qquad\qquad \text{for Sarsa} \end{cases},$$

and γ is the discount factor. The resulting algorithms, called dQ, dSARSA, d$Q(\lambda)$, and dSARSA(λ) are experimentally verified that they performed well for known and constant delay. Among them, they reported that dSARSA(λ) was the most important one.

However, unfortunately, it seems to us that they payed little attention to select next action based on the current observed states. They did not explicitly use delay k for prediction. As we will show in Section 5, if we explicitly predict a sequence of k states by considering the delay, the convergence of learning can be accelerated further.

We now describe our algorithm dSARSA$(\lambda)_k$ in Algorithm 1. Its update rules of $Q(s,a)$ and $e(s,a)$ are based on dSARSA(λ). Note that if $k = 0$, our algorithm dSARSA$(\lambda)_k$ becomes equivalent to the standard Sarsa(λ) using replacing traces with option of clearing the traces of non-selected actions (Sutton and Barto, 1998). Moreover, if changing $Q(\hat{s}_{n+k}, a)$ on line 23 to $Q(s_n, a)$ then it is almost equivalent[1] to dSARSA(λ).

The essential improvement of the algorithm lies in lines 21–23. When the algorithm chooses the next action $a \in A$, it refers $Q(\hat{s}_{n+k}, a)$ instead of $Q(s_n, a)$, where $\hat{s}_{n+k}$ is a predicted state after k steps "simulation" starting from the state s_n. By simulation, we proceed to choose the most likely state at each step. We remark that the same idea has already appeared in the Model Based Simulation algorithm (Walsh et al., 2007).

We implement it as follows. The procedure Memorize(s, a, s') accumulates the number of occurrences of (s, a, s'), the experience that action a in state s yields state s'. By using these numbers, we can simply estimate the probability that the next state becomes s' when taking action a in state s, as

$$\hat{P}(s' \mid s, a) = \frac{\text{the number of occurrences of } (s, a, s')}{\sum_{s' \in S} \text{the number of occurrences of } (s, a, s')}.$$

Then the next state $\hat{s}$ at state s taking action a is predicted by the maximum likelihood principle

$$\hat{s} = \operatorname*{argmax}_{s' \in S} \hat{P}(s' \mid s, a).$$

The procedure Predict$(s_n, \{a_{n-k}, \ldots, a_{n-1}\})$ returns a predicted state $\hat{s}_{n+k}$ after k step starting from s_n, by calculating the following recursive formula

$$\hat{s}_{n+(i+1)} = \operatorname*{argmax}_{s' \in S} \hat{P}(s' \mid \hat{s}_{n+i}, a_{n-(k-i)})$$

for $i = 0, \ldots, k-1$.

4 dSARSA$(\lambda)_X$: ALGORITHM FOR UNKNOWN DELAY

In the previous section, we assumed that the delay was known to the learner. This section considers the case

[1] A subtle difference is the update rule of $e_k(s,a)$ in lines 13–17, although we do not regard it essential.

Algorithm 1: dSARSA(λ)$_k$

Input: learning rate α, discount factor γ, trace-decay parameter λ, action policy π, delay k

1 **Initialize**
2 **for** *any* $s \in S$ *and* $a \in A$ **do**
3 $\quad Q(s,a) \leftarrow 0$;
4 **for** *each episode* **do**
5 $\quad$ **for** *any* $s \in S$ *and* $a \in A$ **do**
6 $\quad\quad e(s,a) \leftarrow 0$;
7 $\quad s_0 \leftarrow$ initial state;
8 $\quad a_0 \leftarrow$ action $a \in A$ selected by π using $Q(s_0,a)$;
9 $\quad$ **for** *each step n of episode* **do**
10 $\quad\quad$ **if** $n \geq k+2$ **then**
11 $\quad\quad\quad$ **for** *any* $s \in S$ *and* $a \in A$ **do**
12 $\quad\quad\quad\quad$ **if** $s = s_{n-2} \wedge a = a_{n-k-2}$ **then**
13 $\quad\quad\quad\quad\quad e(s,a) \leftarrow 1$;
14 $\quad\quad\quad\quad$ **else if** $s = s_{n-2} \wedge a \neq a_{n-k-2}$ **then**
15 $\quad\quad\quad\quad\quad e(s,a) \leftarrow 0$;
16 $\quad\quad\quad\quad$ **else** /* $s \neq s_{n-2}$ */
17 $\quad\quad\quad\quad\quad e(s,a) \leftarrow \gamma \cdot \lambda \cdot e(s,a)$;
18 $\quad\quad\quad$ $\delta \leftarrow r_{n-1} + \gamma \cdot Q(s_{n-1}, a_{n-k-1}) - Q(s_{n-2}, a_{n-k-2})$;
19 $\quad\quad\quad$ **for** *any* $s \in S$ *and* $a \in A$ **do**
20 $\quad\quad\quad\quad Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \delta \cdot e(s,a)$;
21 $\quad\quad\quad$ Memorize(s_{n-1}, a_{n-k-1}, s_n);
22 $\quad\quad\quad$ $\hat{s}_{n+k} \leftarrow$ Predict(s_n,$\{a_{n-k}, \ldots, a_{n-1}\}$);
23 $\quad\quad\quad$ $a_n \leftarrow$ action $a \in A$ selected by π using $Q(\hat{s}_{n+k}, a)$;
24 $\quad\quad$ **else**
25 $\quad\quad\quad$ $a_n \leftarrow$ action $a \in A$ selected by π using $Q(s_n, a)$;
26 $\quad\quad$ Take action a_n, and observe reward r_{n+1} and state s_{n+1};

that the delay is not known, although its maximum value is fixed and known. This is a rational assumption for many practical applications, we believe. For instance, in real-time control problems, if the delay is so long that it can not be recovered by any commands, then nothing helps.

For some special case of UCDMDP, in which an agent has a choice to stay in the same state, and next state is deterministically decided without any noise, the following naive algorithm would succeed to estimate the true value of the delay; after staying in the same state for long time enough, the agent moves only one step, and then keeps staying in the same state. By observing the time stamp of the movement, the agent can easily estimate the delay. However, if the environment is dynamic or under noisy situations, it does not work.

There is an useful property between UCDMDP and dSARSA(λ)$_k$. The property is that if dSARSA(λ)$_k$ is given the real delay, focusing to arguments of Memorize, s_n is random variable generated by the probability distribution depended on only s_{n-1} and a_{n-k-1}, and mutually independent. Thus, it is expected that the prediction performance for s_n by

Algorithm 2: dSARSA(λ)$_X$: Master

Input: learning rate α, discount factor γ, trace-decay parameter λ, action policy π, maximum delay $k_{\max}$

1 **Initialize**
2 **for** $k = 0, \ldots, k_{\max}$ **do**
3 $\quad$ Generate slave with α, γ, λ, π, and k;
4 **for** *each episode* **do**
5 $\quad s_0 \leftarrow$ initial state;
6 $\quad$ **for** *each step n of episode* **do**
7 $\quad\quad$ Recieve the actions and the confidences from all slaves;
8 $\quad\quad$ Take action a_n which is choosed by the slave whose condicence is maximum (ties are broken randomly), and observe reward r_{n+1} and state s_{n+1};
9 $\quad\quad$ Give all slaves a_n, r_{n+1}, and s_{n+1};

Predict(s_{n-1}, $\{a_{n-k-1}\}$) would be high. We propose an algorithm utilizing this property.

Our algorithm has an association with some algorithms for the on-line allocation problem such as Hedge(β) (Freund and Schapire, 1997), and consists of a master and some slaves. The master algorithm, shown in Algorithm 2, has a collection of $k_{\max}+1$ slaves. Each slave is associated with its own value $k \in \{0, \ldots, k_{\max}\}$ as the delay, and works as a slightly modified version of dSARSA(λ)$_k$, which we will explain in detail later. At the end of each step, the master distributes the observation of state and reward to all slaves. Then at the next step, each slave returns a pair of the action and its confidence *conf*. The master simply picks up the action whose confidence is the highest and executes it, and then reports the executed action and distributes the observation of state and reward to all slaves. Based on the feedback, each slave updates its own confidence, as well as $Q(s,a)$ and $e(s,a)$.

We now describe how to get a slave learner from dSARSA(λ)$_k$. Important points are

- Each slave maintains the confidence value *conf* by itself. The confidence is simply a total score of prediction accuracy for the next states.
- Each slave has to update $Q(s,a')$ and $e(s,a')$ for a', where a' is actually selected action by the master, that is not necessarily the one it proposed to the master.

The modifications to the Algorithm 1 are as follows.

1. Insert before line 9:

$$t \leftarrow 0;$$
$$T \leftarrow 0;$$
$$conf_0 \leftarrow 0;$$

to initialize some variables. T is the number of predictions, and t is the number of correct predictions.

2. Insert before line 21:

$$\hat{s}_n \leftarrow Predict(s_{n-1}, \{a_{n-k-1}\});$$
$$\textbf{if } \hat{s}_n = s_n \textbf{ then } t \leftarrow t+1;$$
$$T \leftarrow T+1;$$
$$conf_n \leftarrow \frac{t}{T};$$

3. Replace the line 23 to

$\hat{a} \leftarrow$ action $a \in A$ selected by π using $Q(\hat{s}_{n+k}, a)$;

and replace line 25 to

$\hat{a} \leftarrow$ action $a \in A$ selected by π using $Q(s_n, a)$;

This is because the action $\hat{a}$ that this slave will propose to the master does not necessary equal to the action a_n that the master will actually select.

4. Replace line 26 to

Send action $\hat{a}$ and confidence $conf_n$ to the master.
Receive action a_n, reward r_{n+1}, and state s_{n+1} from the master.

Since all slaves can run in parallel, our algorithm fits to multi-core or multi-processor architectures. We also note that the idea of our algorithm is also applicable to model-based learning (Abbeel et al., 2007; Szita and Szepesvári, 2010).

5 EXPERIMENTS

We now examine dSARSA$(\lambda)_k$ and dSARSA$(\lambda)_X$ for two learning problems with delay, "W-maze" and "cliff edge problem". For comparison, we also examine the following three algorithms.

(1) The standard Sarsa(λ) using replacing traces with clearing the traces of non-selected actions (Sutton and Barto, 1998). In reality, it is equivalent to dSARSA$(\lambda)_k$ with $k = 0$.

(2) dSARSA(λ) proposed by (Schuitema et al., 2010).

(3) MBS+R-max proposed by (Walsh et al., 2007). We note that the algorithm was very slow so that it could not be executed for cliff edge problem.

5.1 W-maze

We first consider the "W-maze" problem illustrated in Figure 1. It was originally introduced in (Walsh et al., 2007), and later extended in (Schuitema et al., 2010). As is often the case with the standard maze problems, the agent can observe its own position as state. Started from a randomly chosen initial position in the maze, the goal of the agent is to reach the cell marked "GOAL", by selecting action among UP, DOWN, RIGHT, and LEFT at each step. If the selected direction is blocked by a wall, the agent stays at the same position. Anyway, the next position is deterministically decided depending on the current position and the selected action. The agent suffers a negative reward of -1 at each step.

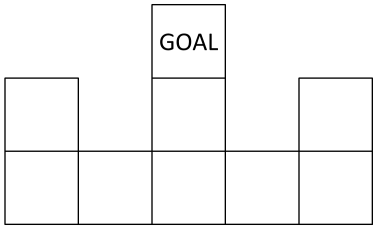

Figure 1: W-maze (Walsh et al., 2007).

At first, we determined several meta parameters of learning in non-delayed situation, so that they can learn efficiently. We chose that the learning rate $\alpha = 1.0$, discount factor $\gamma = 0.5$, trace-decay parameter $\lambda = 0.5$ for all Sarsa(λ), dSARSA(λ), dSARSA$(\lambda)_k$ and dSARSA$(\lambda)_X$. As action policies, we selected the *greedy action policy without exploration* for *Sarsa*(λ), dSARSA$(\lambda)_k$ and dSARSA$(\lambda)_X$, but not for dSARSA(λ). Because we had observed that dSARSA(λ) worked quite badly if we combined it with the greedy action policy by prior experiments. Therefore, for dSARSA(λ), we selected *softmax action selection* (Sutton and Barto, 1998), and chose the inverse temperature $\beta = 0.1$. Moreover, we also execute dSARSA(λ) with the learning rate $\alpha = 0.1$, since (Schuitema et al., 2010) noted that the learning rate α for dSARSA(λ) should be small. For MBS+R-max, meta parameters were selected to be $T = 10$ and $K_1 = 1$, which were actually the meta parameters for R-max (Brafman and Tennenholtz, 2003).

In the experiments, the delay k was 5, and $k_{\max}$ for dSARSA$(\lambda)_X$ was 15. One episode consists of the steps from initial position to the goal, and we executed 100 episodes for 10 times, in order to calculate the average of accumulated rewards for each episode.

5.1.1 Noiseless Environment

We show the results in Figure 2 and Figure 3. For comparison, we additionally plotted the averaged accumulated reward of Sarsa(λ) for the problem with no delay, that should be regarded as the ideal accuracy. In Figure 2, we observe that both MBS+R-max

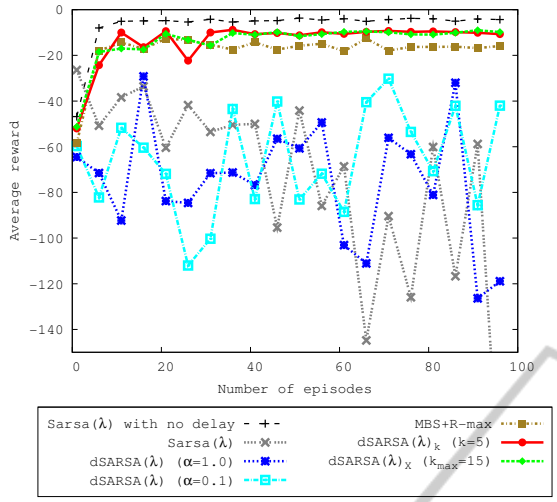

Figure 2: Comparison of Sarsa(λ) with no delay (for reference), Sarsa(λ), dSARSA(λ), MBS+R-max, dSARSA(λ)$_k$, and dSARSA(λ)$_X$, on W-maze.

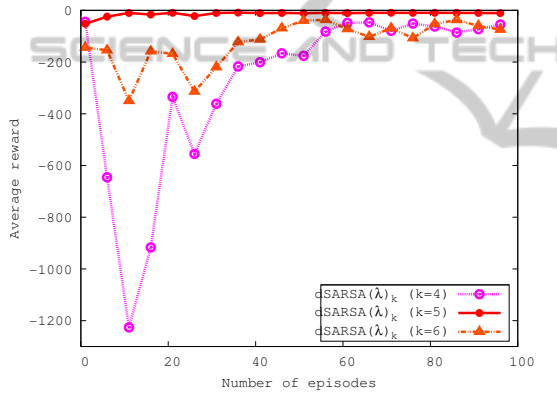

Figure 3: dSARSA(λ)$_k$ for $k = 5$ (true delay), compared with $k = 4$ and 6 (wrong delays), on W-maze.

and dSARSA(λ)$_k$ given the real delay ($k = 5$) performed as good as the ideal, Sarsa(λ) with no delay, while dSARSA(λ) did not. It implies that the prediction in dSARSA(λ)$_k$ is indeed effective to accelerate the learning.

Next, we verified the performance of dSARSA(λ)$_k$ with varying k. For the same problem, we executed dSARSA(λ)$_k$ with $k = 4$, 5, and 6, as shown in Figure 3. It is clear that dSARSA(λ)$_k$ with wrong value ($k \neq 5$) converges very slowly, compared to the one with true value $k = 5$.

Let us turn our attentions to dSARSA(λ)$_X$. Figure 2 shows that dSARSA(λ)$_X$ learns as quickly as dSARSA(λ)$_k$ with true value $k = 5$, although dSARSA(λ)$_X$ does not know the true value. We then examine how the parameter $k_{\max}$ affects to the performance of dSARSA(λ)$_X$. Figure 4 shows the average and standard deviation of the predicted delay by

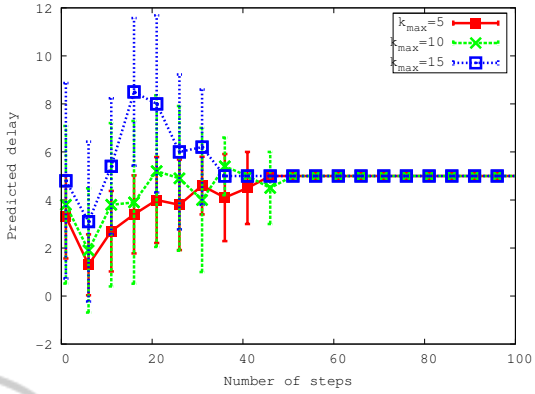

Figure 4: predicted delay by dSARSA(λ)$_X$ with varying $k_{\max}$ on W-maze. We plotted the average and standard deviation for every 5 steps.

dSARSA(λ)$_X$ with $k_{\max} \in \{5, 10, 15\}$, for 10 times. X-axis is the number of steps which each algorithm runs. We see that for any upper bound $k_{\max} = 5$, 10, and 15, predicted delay value converses to the true value $k = 5$ before 60 steps. Recall that in Figure 2, dSARSA(λ)$_X$ with $k_{\max} = 15$ received the reward -51.3 on average at the first episode. It means that at the first episode, it consumed 51.3 steps on average. Moreover, we verified that at the first episode, dSARSA(λ)$_X$ consumed 43.7 steps for $k_{\max} = 10$, and 52.0 steps for $k_{\max} = 5$ on average, although we omitted to draw them in the graph. These results imply that the estimation of the true delay finished before the first episode ends, for any of $k_{\max} = 5$, 10, and 15.

5.1.2 Noisy Environment

We also tried the same problem in noisy environment. Here, each action succeeds with probability of 0.7, and otherwise, one of the other three directions is randomly chosen with probability 0.1 for each. In such a noisy environment, it should be difficult to keep staying in the same state, so that we cannot apply the naive algorithm mentioned above.

We determined the meta parameters of learning as follows. Learning rate $\alpha = 0.1$, the discount factor $\gamma = 0.5$, trace-decay parameter $\lambda = 0.5$ for all Sarsa(λ), dSARSA(λ), dSARSA(λ)$_k$ and dSARSA(λ)$_X$. Additionally for dSARSA(λ), we also executed it with learning rate $\alpha = 0.01$. As action policies, we chose the *greedy action selection* for all algorithms. For MBS+R-max, we set $T = 10$ and $K_1 = 10$. The delay $k = 5$ and $k_{\max} = 5$ for dSARSA(λ)$_X$.

Figure 5, Figure 6, and Figure 7 show the results in noisy environments, each of which corresponds to Figure 2, Figure 3, and Figure 4 in noiseless environment, respectively. We can verify that the problem is

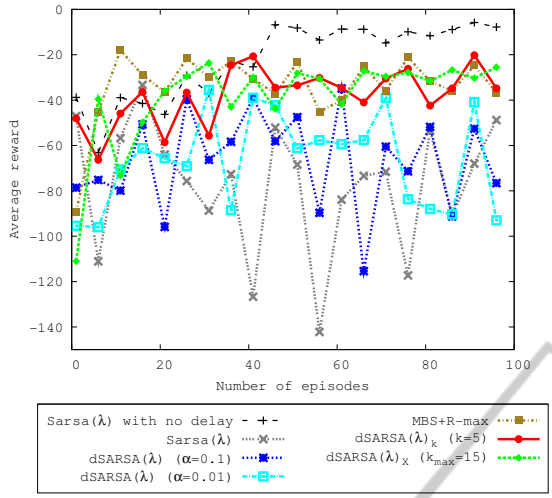

Figure 5: Comparison of Sarsa(λ) with no delay (for reference), Sarsa(λ), dSARSA(λ), MBS+R-max, dSARSA(λ)$_k$, and dSARSA(λ)$_X$, on *Noisy* W-maze.

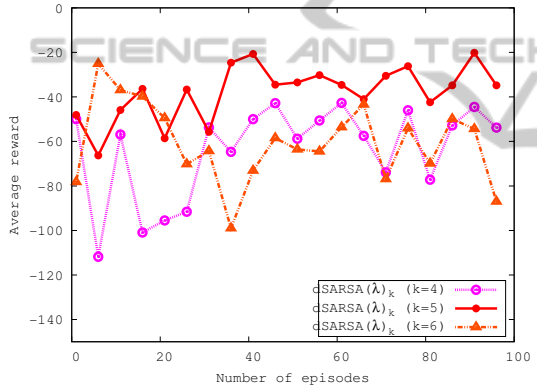

Figure 6: dSARSA(λ)$_k$ for $k = 5$ (true delay), compared with $k = 4$ and 6 (wrong delays), on *Noisy* W-maze.

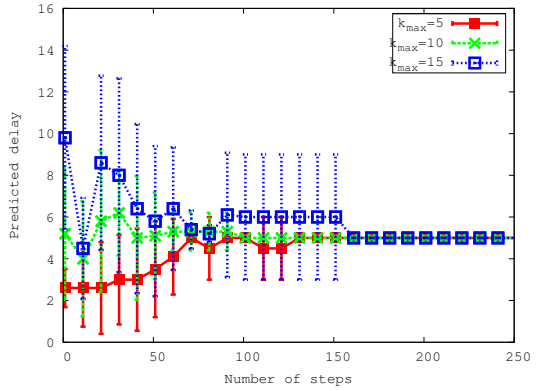

Figure 7: Predicted delay by dSARSA(λ)$_X$ with varying $k_{\max}$ on *Noisy* W-maze. We plotted the average and standard deviation for every 10 steps.

indeed more difficult than the problem with no delay; the learning is slow and the total reward is smaller than Sarsa(λ) with no delay, because of the strong noise. However, if we turn our attention to the algorithms for known delay, dSARSA(λ)$_k$ is more efficient than the other algorithms. We also see that dSARSA(λ)$_X$ performs as good as dSARSA(λ)$_k$, although the former does not know the delay while the latter knows it. Moreover, although the problem cannot be solved by the naive algorithm , dSARSA(λ)$_X$ succeeded to estimate the delay accurately; estimation finished before the first three episodes end.

5.2 Cliff Edge Problem

We propose a new problem, named *cliff edge problem* which is illustrated in Figure 8. Imagine the situation that an agent approaches to the cliff edge at the right ende. The nearer to the cliff edge the agent stands, the higher rewards it gets. However, it approaches too nearly, it falls down. Formally, the agent can observe its own position as state $s_1, \ldots, s_h$. Started from the initial position s_1, the agent selects an action among LEFT, STAY, and RIGHT at each step, and the next state is decided deterministically. The agent gets a reward of $+i$ when agent is in state s_i. However, if the agent tries to move RIGHT at state s_h, it returns to the leftmost (initial) position.

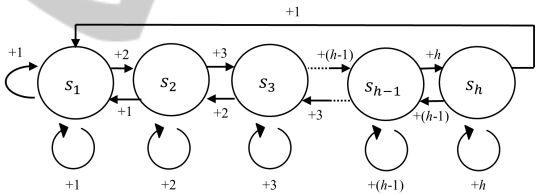

Figure 8: Cliff edge problem: Agent selects a action among LEFT, STAY, RIGHT at each state in $\{s_1, \ldots, s_h\}$. Agent get a reward of $+i$ when agent is in state s_i.

The problem would be easy with no delay under noiseless environment. However, if we consider the delay and/or noise, it would be considerably difficult, since inaccurate observation of the state is fatal to the agent.

We determined the meta parameters in the same way as the previous experiment. The learning rate $\alpha = 0.3$, the discount factor $\gamma = 0.2$, trace-decay parameter $\lambda = 0.4$ for Sarsa(λ), dSARSA(λ), dSARSA(λ)$_k$ and dSARSA(λ)$_X$. Moreover, we also examined dSARSA(λ) with the learning rate $\alpha = 0.03$. We chose the *softmax action selection* as action policy for Sarsa(λ), dSARSA(λ)$_k$, dSARSA(λ)$_X$, and let the inverse temperature β be 0.2. The delay $k = 5$, and the uppper bound of the delay $k_{\max} = 15$ for dSARSA(λ)$_X$. We executed 3,000 steps from the initial position as one episode. We repeated it for 10

times and calculated the average of reward at each step.

5.2.1 Noiseless Environment

The results in noiseless environment are shown in Figure 9 and Figure 10. For comparison, we additionally plotted the averaged reward of Sarsa(λ) for the problem with no delay. Furthermore, we show the average and standard deviation of 10 times for the prediction of state by dSARSA$(\lambda)_X$ with $k_{\max} \in \{5, 10, 15\}$ in Figure 11.

These results have almost the same tendency to the previous experiments on "W-maze". If the delay k is known correctly, dSARSA$(\lambda)_k$ performs better than the other methods. For unknown k, dSARSA$(\lambda)_X$ converges to the real value quickly.

5.2.2 Noisy Environment

We also tried the same problem "cliff edge" in noisy environment, where the agent moves to the desired direction with probability 0.9, but moves randomly to one of the other two directions with probability 0.05 for each. We chose the meta parameters as follows, based on prior experiments. The learning rate $\alpha = 0.05$, the discount factor $\gamma = 0.2$, trace-decay parameter $\lambda = 0.4$ for Sarsa(λ), dSARSA(λ), dSARSA$(\lambda)_k$ and dSARSA$(\lambda)_X$. Additionally for dSARSA(λ), we also execute it with the learning rate $\alpha = 0.005$. As action policies, we chose the *softmax action selection* for Sarsa(λ), dSARSA$(\lambda)_k$, and dSARSA$(\lambda)_X$, and let the inverse temperature β be 0.5. The delay $k = 5$ and the upper bound $k_{\max} = 15$ for dSARSA$(\lambda)_X$. Figure 12, Figure 13, and Figure 14 in noisy environments corresponds to Figure 9, Figure 10, and Figure 11 in noiseless environments, respectively.

Because of the noise, the learning task became significantly difficult. However, efficiency of proposed algorithms is as same as the Sarsa(λ) with no delay, and convergence of prediction of state is also fast.

6 CONCLUSIONS

In this paper, we dealt with the reinforcement learning for environments with control delay. We proposed dSARSA$(\lambda)_k$ that improved dSARSA(λ) by predicting current states, which works for known delay. We verified that dSARSA$(\lambda)_k$ performs as accurate as MBS+R-max, which is one of model-based learning algorithms requiring much more computation resources, while dSARSA(λ) did not work well.

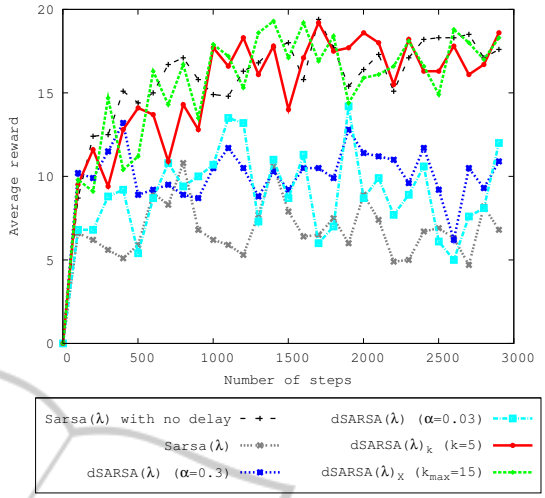

Figure 9: Comparison of Sarsa(λ) with no delay (for reference), Sarsa(λ), dSARSA(λ), dSARSA$(\lambda)_k$, and dSARSA$(\lambda)_X$, on Cliff edge problem.

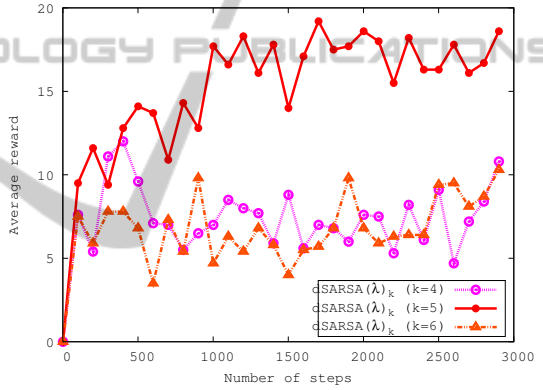

Figure 10: dSARSA$(\lambda)_k$ for $k = 5$ (true delay), compared with $k = 4$ and 6 (wrong delays), on Cliff edge problem.

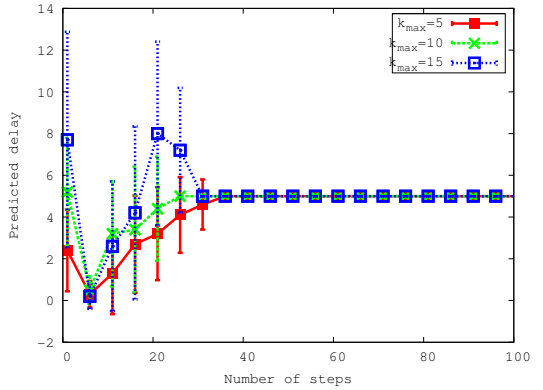

Figure 11: Predicted delay by dSARSA$(\lambda)_X$ with varying $k_{\max}$ on Cliff edge problem. We plotted the average and standard deviation for every 5 steps.

For the case that the delay is unknown, we proposed dSARSA$(\lambda)_X$, that combines sev-

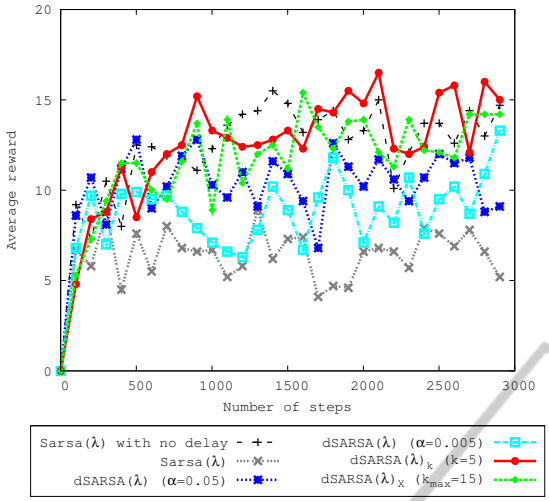

Figure 12: Comparison of Sarsa(λ) with no delay (for reference), Sarsa(λ), dSARSA(λ), dSARSA$(\lambda)_k$, and dSARSA$(\lambda)_X$, on *Noisy* Cliff edge problem.

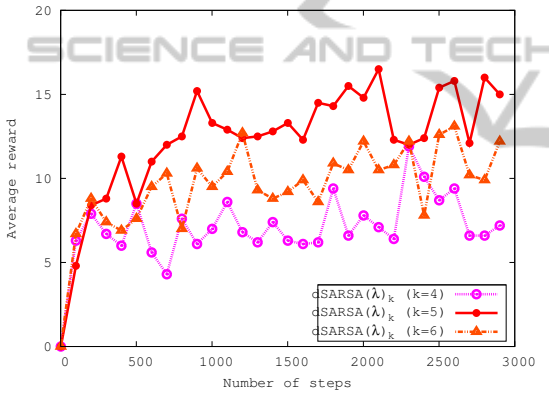

Figure 13: dSARSA$(\lambda)_k$ for $k = 5$ (true delay), compared with $k = 4$ and 6 (wrong delays), on *Noisy* Cliff edge problem.

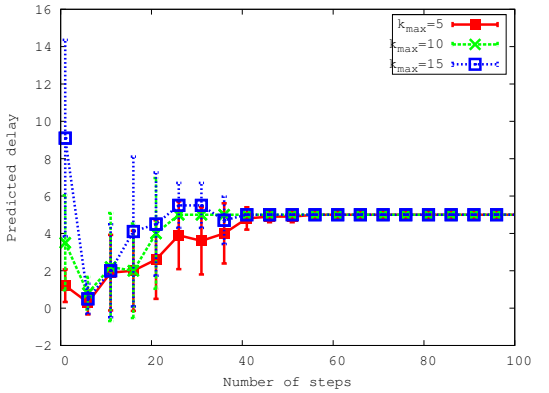

Figure 14: Predicted delay by dSARSA$(\lambda)_X$ with varying $k_{\max}$ on *Noisy* Cliff edge problem. We plotted the average and standard deviation for every 5 steps.

eral dSARSA$(\lambda)_k$'s as slaves. We confirmed that dSARSA$(\lambda)_X$ performs competitively as the algorithms given the real delay.

As future work, we are interested in theoretical analysis and expanding for continuous states, actions and time, as well as applications to real environments.

ACKNOWLEDGEMENTS

This work was partially supported by Kakenhi 23300051.

REFERENCES

Abbeel, P., Coates, A., Qugley, M., and Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. In *In Advances in Neural Information Processing Systems 19*, pages 1–8.

Brafman, R. I. and Tennenholtz, M. (2003). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231.

Freund, Y. and Schapire, R. (1997). A desicion-theoretic generalization of on-line learning and an application to boosting. volume 55, pages 119–139.

Katsikopoulos, K. and Engelbrecht, S. (2003). Markov decision processes with delays and asynchronous cost collection. *IEEE Transactions on Automatic Control*, 48(4):568–574.

Loch, J. and Singh, S. (1998). Using eligibility traces to find the best memoryless policy in partially observable markov decision processes. In *Proceedings of the 15th International Conference on Machine Learning (ICML '98)*, pages 323–331.

Schuitema, E., Busoniu, L., Babuška, R., and Jonker, P. (2010). Control delay in reinforcement learning for real-time dynamic systems: a memoryless approach. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3226–3231.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press.

Szita, I. and Szepesvári, C. (2010). Model-based reinforcement learning with nearly tight exploration complexity bounds. In *Proceedings of the 27th International Conference on Machine Learning (ICML '10)*, pages 1031–1038.

Walsh, T. J., Nouri, A., Li, L., and L.Littman, M. (2007). Planning and learning in environments with delayed feedback. In *Proceedings of the 18th European Conference on Machine Learning (ECML '07)*, pages 442–453.