Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 238357, 8 pages
http://dx.doi.org/10.1155/2014/238357

Research Article

Intelligent Inventory Control via Ruminative Reinforcement Learning

Tatpong Katanyukul[1] and Edwin K. P. Chong[2]

[1] *Faculty of Engineering, Khon Kaen University, Computer Engineering Building, 123 Moo 16, Mitraparb Road, Muang, Khon Kaen 40002, Thailand*
[2] *Department of Electrical and Computer Engineering, Colorado State University, 1373 Campus Delivery, Fort Collins, CO 80523-1373, USA*

Correspondence should be addressed to Tatpong Katanyukul; tatpong@gmail.com

Received 30 December 2013; Accepted 29 May 2014; Published 7 July 2014

Academic Editor: Aderemi Oluyinka Adewumi

Inventory management is a sequential decision problem that can be solved with reinforcement learning (RL). Although RL in its conventional form does not require domain knowledge, exploiting such knowledge of problem structure, usually available in inventory management, can be beneficial to improving the learning quality and speed of RL. Ruminative reinforcement learning (RRL) has been introduced recently based on this approach. RRL is motivated by how humans contemplate the consequences of their actions in trying to learn how to make a better decision. This study further investigates the issues of RRL and proposes new RRL methods applied to inventory management. Our investigation provides insight into different RRL characteristics, and our experimental results show the viability of the new methods.

1. Introduction

Inventory management is a crucial business activity and can be modeled as a sequential decision problem. Bertsimas and Thiele [1], among others, addressed the need for an efficient and flexible inventory solution that is also simple to implement in practice. This may be among the reasons for extensive studies of reinforcement learning (RL) application to inventory management.

RL [2, 3] is an approach to solve sequential decision problems based on learning the underlying state value or state-action value. Relying on learning mechanism, RL in its typical form does not require knowledge of a structure of the problem. Therefore, RL has been studied in wide range of sequential decision problems, for example, virtual machine configuration [4], robotics [5], helicopter control [6], ventilation, heating and air conditioning control [7], electricity trade [8], financial management [9], water resource management [10], and inventory management [11]. Acceptance of RL is credited to RL's effectiveness, potential possibilities [12], link to mammal learning processes [13, 14], and its model-free property [15].

Despite fascination with RL's model-free property, most inventory management problems can naturally be formulated into a well-structured part interacting with another part that is less understood. That is, replenishment cost, holding cost, and penalty cost can be determined precisely in advance. On the other hand, customer demand or, in some cases, delivery time or availability of supplies is usually less predictable. However, once a value of a less predictable variable is known, the period cost can be determined precisely. Specifically, a warehouse would know its period inventory cost after its replenishment has arrived and all demand orders in the period have been observed. Calculation of a period cost is a well-defined formula, while another part, for example, demand, is less predictable. Knowledge about the well-structured part can be exploited, while a learning mechanism can be used to handle the less understood part.

Utilizing this knowledge, Kim et al. [16] proposed *asynchronous action-reward learning*, which used simulation to

evaluate consequences of actions not taken in order to accelerate the learning process in a stateless system. Extending the idea to state-based system, Katanyukul [17] developed ruminative reinforcement learning (RRL) methods, that is, ruminative SARSA (RSarsa) and policy-weighted RSarsa (PRS). The RRL approach is motivated by how humans contemplate consequences of their actions to improve their learning hoping to make a better decision. His study of RRL reveals good potential of the approach. However, existing individual methods show strengths in different scenarios: RSarsa is shown to have fast learning but leads to inferior learning quality in a long-term run. PRS is shown to lead to superior learning quality in a long-term run, but with slower rate.

Our proposed method here is developed to exploit the fast learning characteristic of RSarsa and good learning quality in a long-term run of PRS. Our experimental results show effectiveness of the proposed method and support our assumption underlying development of RRL.

2. Background

An objective of a sequential inventory management is to minimize a long-term cost, $C_0(s_0) = \min_{a_0,\ldots,a_T} \sum_{t=0}^{T} \gamma^t E[c_t \mid s_0, a_0, \ldots, a_T]$, subject to $a_t \in A_{s_t}$ for $t = 0, \ldots, T$, where $E[c_t \mid s_0, a_0, \ldots, a_T]$ is the expected period cost of period t given an initial state s_0 and actions $a_0, \ldots, a_T$ over periods 0 to T, respectively; γ is a discount factor; and A_{s_t} is a feasible action set at state s_t. Under certain assumptions, the problem can be posed as a Markov decision problem (MDP) (see [15] for details). In this case, what we seek is an *optimal policy*, which maps each state to an optimal action. Given an arbitrary policy π, the long-term state cost for that policy can be written as

$$C^{\pi}(s) = r^{\pi}(s) + \gamma \sum_{s'} p^{\pi}\left(s' \mid s\right) C^{\pi}\left(s'\right), \tag{1}$$

where $r^{\pi}(s)$ is an expected period state cost, $p^{\pi}(s' \mid s)$ is a transition probability—the probability of the next state being s' when the current state is s. The superscript π notation indicates dependence on the policy π. In practice, exact solution to (1) is difficult to find. Reinforcement learning (RL) [2] provides a framework to find an approximate solution. An approximate long-term cost of state s is obtained by summation of the period cost and a long-term cost of the next state $Q(s) = r(s) + \gamma Q(s')$.

The RL approach is based on temporal difference (TD) learning, which uses temporal difference error ψ (2) to estimate the long-term cost (3):

$$\psi = r + \gamma Q\left(s', a'\right) - Q(s, a) \tag{2}$$

$$Q^{(\text{new})}(s, a) = Q^{(\text{old})}(s, a) + \alpha \cdot \psi, \tag{3}$$

where r is the period cost, which corresponds to taking action a in state s, α is a learning rate, and s' and a' are the state and action taken in the next period, respectively.

Once the values of $Q(s, a)$ are thoroughly learned, they are good approximations of long-term costs. We often refer to $Q(s, a)$ as the "Q-value." Most RL methods determine the actions to take based on Q-values. These methods include SARSA [2], a widely used RL algorithm. We use SARSA as a benchmark, representing a conventional RL method, to compare with other methods under investigation. In each period, given observed state s, action taken a, observed period cost r, observed next state s', and anticipating next action taken a', the SARSA algorithm updates the Q-value based on TD learning (2) and (3).

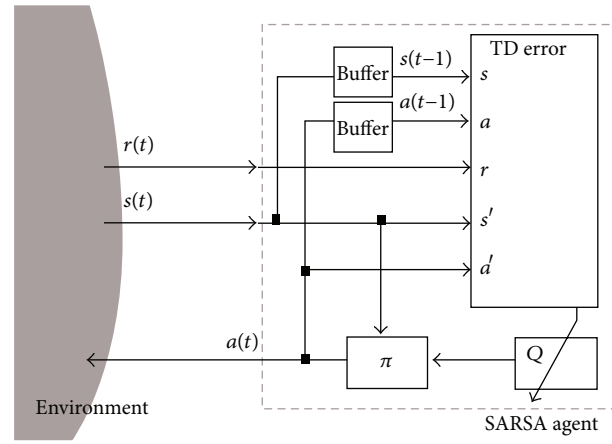

Figure 1: SARSA agent and interacting variables (this figure is adapted from Figure 6.15 of Sutton and Barto [2]).

Based on the Q-value, we can define a policy π to determine an action to take at each state. The policy is usually stochastic, defined by a probability $p(a \mid s)$ to take an action a given a state s. The policy has to balance between taking the best action based on the currently learned Q-value and trying another alternative. Trying another alternative gives the learning agent a chance to explore thoroughly the consequences of its state-action space. This helps to create a constructive cycle of improving the quality of learned Q-values, which in turn will help the agent to choose better actions and reduce the chance to get stuck in a local optimum. This is an issue of balancing between *exploitation* and *exploration*, as discussed in Sutton and Barto [2]. (Since the RL algorithm is autonomous and interacts with its environment, we sometimes use the term "learning agent.").

An ϵ-greedy policy is a general RL policy, which also is easy to implement. With probability ϵ, the policy chooses an action randomly from $a \in A(s)$, where $A(s)$ is a set of allowable actions given state s. Otherwise, it takes an action corresponding to the minimal current Q-value, $a^* = \arg\min_a Q(s, a)$.

3. Ruminative Reinforcement Learning

The conventional RL approach, SARSA, assumes that the agent knows only the current state s, the action a it takes, the period cost r, the next state s', and the action a' it will take in the next state. Each period, the SARSA agent updates the Q-value based on the TD error calculated with these five variables. Figure 1 illustrates the SARSA agent, the five variables it needs to update the Q-value, and its interaction with its environment.

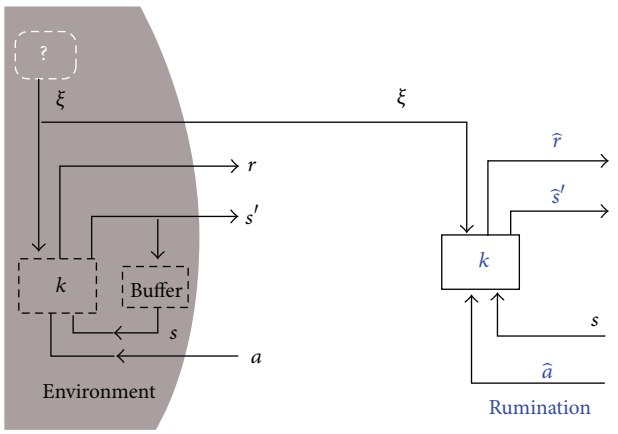

FIGURE 2: Environment, knowledge of its structure, and rumination.

(L00) Initialize $Q(s, a)$.
(L01) Observe s.
(L02) Determine a by policy π.
(L03) For each period,
(L04) observe r, s', and ξ;
(L05) determine a' by policy π;
(L06) calculate $\psi = r + \gamma Q(s', a') - Q(s, a)$;
(L07) update $Q(s, a) \leftarrow Q(s, a) + \alpha \cdot \psi$;
(L08) for each $\hat{a} \in \widehat{A}(s)$,
(L09) calculate $\hat{r}, \hat{s}'$ with $k(s, \hat{a}, \xi)$,
(L10) determine $\hat{a}'$,
(L11) calculate $\psi = \hat{r} + \gamma Q(\hat{s}', \hat{a}') - Q(s, \hat{a})$,
(L12) update $Q(s, \hat{a}) \leftarrow Q(s, \hat{a}) + \alpha \cdot \psi$
(L13) until ruminated all $\hat{a} \in \widehat{A}$;
(L14) set $s \leftarrow s'$ and $a \leftarrow a'$
(L15) until termination.

ALGORITHM 1: RSarsa algorithm.

However, in inventory management problems, we usually have extra knowledge about the environment. That is, the problem structure can naturally be formulated such that the period cost r and next state s' are determined by a function $k : s, a, \xi \mapsto r, s'$, where ξ is an extra information variable. This variable ξ captures the stochastic aspect of the problem. The process generating ξ may be unknown, but the value of ξ is fully observable after the period is over. Given a value of ξ, along with s and a, the deterministic function k can precisely determine r and s'.

Without this extra knowledge, each period, the SARSA agent updates only one value of $Q(s, a)$ corresponding to current state s and action taken a. However, with the function k and an observed value of ξ, we can do "rumination": evaluating the consequences of other actions $\hat{a}$, even those that were not taken. Figure 2 illustrates rumination and its associated variables. Given the rumination mechanism, we can provide information required by SARSA's TD calculation for any underlying action. Katanyukul [17] introduced this rumination idea and incorporated it into the SARSA algorithm, resulting in the *ruminative SARSA (RSarsa)* algorithm. Algorithm 1 shows the RSarsa algorithm. It should be noted that RSarsa is similar to SARSA, but with inclusion of rumination from line 8 to line 13.

The experiments in [17] showed that RSarsa had performed significantly better than SARSA in early periods (indicating faster learning), but its performance was inferior to SARSA in later periods (indicating poor convergence to the appropriate long-term state cost approximation). Katanyukul [17] attributed RSarsa's poor long-term learning quality to its lack of natural action visitation frequency.

TD learning (2) and (3) update the Q-value as an approximation of the long-term state cost. The transition probability $p^{\pi}(s' \mid s)$ in (1) does not appear explicitly in the TD learning calculation. Conventional RL relies on sampling trajectories to reflect the natural frequency of visits to state-action pairs corresponding to the transition probability. It updates only the state-action pairs as they are actually visited; therefore, it does not require explicit calculation of the transition probability and still eventually converges to a good approximation.

However, because RSarsa does rumination for all actions ignoring their sampling frequency, this is equivalent to disregarding the transition probability, which leads to RSarsa's poor long-term learning quality.

To address this issue, Katanyukul [17] proposed *policy-weighted RSarsa (PRS)*. PRS explicitly calculates probabilities of actions to be ruminated and adjusts the weights of their updates. PRS is similar to RSarsa, but the rumination update (line 12 in Algorithm 1) is replaced by

$$Q(s, \hat{a}) \longleftarrow Q(s, \hat{a}) + \beta \cdot \psi, \tag{4}$$

where $\beta = \alpha \cdot p(\hat{a})$ and $p(\hat{a})$ is the probability of taking action $\hat{a}$ in state s with policy π. Given an ϵ-greedy policy, we have $p(\hat{a}) = \epsilon / |\widehat{A}(s)|$ for $\hat{a} \neq a^*$ and $p(\hat{a}) = \epsilon / |\widehat{A}(s)| + (1 - \epsilon)$ otherwise, where $|\widehat{A}(s)|$ is a number of allowable actions. PRS has been shown to perform well in early and later periods, compared to SARSA. However, RSarsa is reported to significantly outperform PRS in early periods.

4. New Methods

According to the results of [17], although RSarsa may converge to a wrong approximation, RSarsa was shown to perform impressively in the very early periods. This suggests that if we jump-start the learning agent with RSarsa and then later switch to PRS, before the Q-values settle into bad spots, we may be able to achieve both faster learning and good approximation for a long-term run.

PRS.Beta. We first introduce a straightforward idea, called PRS.Beta, where we will use a varying ruminative learning rate as a mechanism to shift from full rumination (RSarsa) to policy-weighted rumination (PRS). Similar to PRS, the rumination update is determined by (4). However, the value of the rumination learning rate β is determined by

$$\beta = \alpha \cdot \{1 - (1 - p(\hat{a})) \cdot f\}, \tag{5}$$

where f is a function having a value between 0 and 1. When $f \to 0$, $\beta \to \alpha$ and the algorithm will behave like RSarsa.

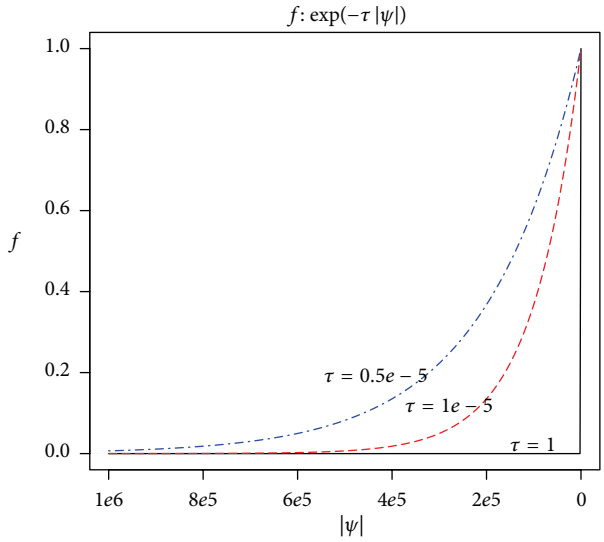

FIGURE 3: Function $\exp(-\tau|\psi|)$ and effects of different τ values.

When $f \to 1$, $\beta \to \alpha \cdot p(\widehat{a})$ and the algorithm will behave like PRS. We want f to start out close to 0 and grow to 1 at a proper rate. By examining our preliminary experiments, the TD error will get smaller as the learning converges. This is actually a property of TD learning. Given this property, we can use the magnitude of the TD error $|\psi|$ to control the shifting, such that

$$f(\psi) = \exp\left(-\tau \cdot |\psi|\right), \tag{6}$$

where τ is a scaling factor. Figure 3 illustrates the effects of different values of τ. Since the magnitude of τ should be relative to $|\psi|$, we set $\tau = |2/(r + Q(s,a))|$, so that the magnitude of τ will be in a proper scale relative to $|\psi|$ and automatically adjusted.

RSarsa.TD. Building on the PRS.Beta method above, we next propose another method, called RSarsa.TD. The underlying idea is that since SARSA performs well in a long-term run (see [2] for theoretical discussion of SARSA's optimality and convergence properties), then after we speed up the early learning process with rumination, we can just switch back to SARSA. This approach is to utilize the fast learning characteristic of full rumination in early periods and to avoid its poor long-term performance. In addition, as a computational cost of rumination is proportional to the size of the ruminative action space $|\widehat{A}(s)|$, this also helps to reduce the computational cost incurred by rumination. It is also intuitively appealing in the sense that we do rumination only when we need it.

The intuition to selectively do rumination was introduced in [17] in an attempt to reduce the extra computational cost from rumination. There, the probability to do rumination was a function of the magnitude of the TD error:

$$p(\text{rumination}) = 1 - \exp\left(-\left|\frac{2\psi}{r + Q(s,a)}\right|\right). \tag{7}$$

However, Katanyukul [17] investigated this selective rumination only with the policy-weighted method and called it PRS.TD. Although PRS.TD was able to improve the computational cost of the rumination approach, the inventory management performance of PRS.TD was reported to have mixed results, implying that incorporation of selective rumination may deteriorate performance of PRS.

This performance deterioration may be due to using p(rumination) with policy weighted correction. Both schemes use $|\psi|$ to control their effect of rumination; therefore, they might have an effect equivalent to overcorrecting the state-transition probability. Unlike PRS, RSarsa does not correct the state-transition probability. Incorporating selective rumination (7) will be the only scheme controlling rumination with $|\psi|$. Therefore, we expect that this approach may allow the advantage of RSarsa's fast learning, while maintaining the long-term learning quality of SARSA.

5. Experiments and Results

Our study uses computer simulations to conduct numerical experiments on three inventory management problem settings (P1, P2, and P3). All problems are periodic review single-echelon with nonzero setup cost. P1 and P2 have one-period lead time. P3 has two-period lead time. The same Markov model is used to govern all problem environments, but with different settings. For P1 and P2, the problem state space is $\mathbb{I} \times \{0, \mathbb{I}^+\}$, for on-hand and in-transit inventories: x and $b^{(1)}$, respectively. P3's state space is $\mathbb{I} \times \{0, \mathbb{I}^+\} \times \{0, \mathbb{I}^+\}$, for x and in-transit inventories $b^{(1)}$ and $b^{(2)}$. The action space is $\{0, \mathbb{I}^+\}$, for replenishment order a.

The state transition is specified by

$$\begin{aligned} x_{t+1} &= x_t + b_t^{(1)} - d_t, \\ b_{t+1}^{(i)} &= b_t^{(i+1)}, \quad \text{for } i = 1, \ldots, L-1, \\ b_{t+1}^{(L)} &= a_t, \end{aligned} \tag{8}$$

where L is a number of lead time periods.

The inventory period cost is calculated from the equation

$$\begin{aligned} r_t = {} & K \cdot \delta(a_t) + G \cdot a_t + H \cdot x_{t+1} \cdot \delta(x_{t+1}) \\ & - B \cdot x_{t+1} \cdot \delta(-x_{t+1}), \end{aligned} \tag{9}$$

where K, G, H, and B are setup, unit, holding, and penalty costs, respectively, and $\delta(\cdot)$ is a step function. Five RL agents are studied: SARSA, RSarsa, PRS, RSarsa.TD, and PRS.Beta.

Each experiment is repeated 10 times. In each repetition, an agent is initialized with all zero Q-values. Then, the experiment is run consecutively for N_E episodes. Each episode starts with initial state and action as follows: for all problems, $b_1^{(1)}$ and a_1 are initialized with values randomly drawn between 0 and 100. In P1, x_1 is initialized to 50; in P2, x_1 is initialized from randomly drawn values between −50 and 400; in P3, x_1 is initialized to 100 and randomly drawn values of $b^{(2)}$ between 0 and 100. Each episode ends when N_P periods are reached or an agent has visited a termination

Table 1: Experimental results.

Line		Methods				
		SARSA	RSarsa	PRS	RSarsa.TD	PRS.Beta
		Relative computation time/epoch				
1	P1	1	30	26	5	30
2	P2	1	20	21	3	19
3	P3	1	31	29	6	31
		Average cost of early periods				
4	P1	8,421	7,619 (W)	8,379 (p0.43)	7,597 (W)	7,450 (W)
5	P2	4,935	4,606 (W)	4,792 (p0.06)	4,685 (W)	4,411 (W)
6	P3	10,502	8,694 (W)	9,958 (p0.20)	9,390 (p0.07)	8,472 (W)
		Average cost of later periods				
7	P1	7,214	7,355 (p0.68)	7,051 (W)	7,110 (p0.11)	7,010 (W)
8	P2	4,308	4,388 (p0.90)	4,248 (p0.14)	4,375 (p0.84)	4,194 (W)
9	P3	8,613	8,139 (p0.29)	8,312 (p0.37)	8,486 (p0.43)	7,664 (p0.18)

state, which is a state lying outside a valid range of Q-value implementation. The maximum number of periods in each episode, N_P, defines the length of the problem horizon, while the number of episodes N_E specifies a variety of problem scenarios, that is, different initial states and actions.

Three problem settings are used in our experiments. Problem 1 (P1) has $N_E = 100$, $N_P = 60$, $K = 200$, $G = 100$, $B = 200$, and $H = 20$. Demand d_t is normally distributed, with mean 50 and standard deviation 10, denoted as $d_t \sim \mathcal{N}(50, 10^2)$. The environment state $[x, b^{(1)}]$ is set as the RL agent state $s = [x, b^{(1)}]$. Problem 2 (P2) has $N_E = 500$, $N_P = 60$, $K = 200$, $G = 50$, $B = 200$, and $H = 20$, with demand $d_t \sim \mathcal{N}(50, 10^2)$. The RL agent state is set as the inventory level $s = x + b^{(1)}$. Therefore, the RL agent state is one-dimensional. Problem 3 (P3) has $N_E = 500$, $N_P = 60$, $K = 200$, $G = 50$, $B = 200$, and $H = 20$. The demand d_t is AR1/GARCH(1,1): $d_t = a_0 + a_1 \cdot d_{t-1} + \epsilon_t$; $\epsilon_t = e_t \cdot \sigma_t$ and $\sigma_t^2 = \nu_0 + \nu_1 \cdot \epsilon_{t-1}^2 + \nu_2 \cdot \sigma_{t-1}^2$, where a_0 and a_1 are AR1 model parameters; ν_0, ν_1, and ν_2 are GARCH(1,1) parameters; and e_t is white noise distributed according to $\mathcal{N}(0, 1)$. The values of AR1/GARCH(1,1) in our experiments are $a_0 = 2$, $a_1 = 0.8$, $\nu_0 = 100$, $\nu_1 = 0.1$, and $\nu_2 = 0.8$, with initial values $d_1 = 50$, $\sigma_1^2 = 100$, and $\epsilon_1 = 2$. The RL agent state in P3 is three-dimensional $s = [x, b^{(1)}, b^{(2)}]$. In all three problem settings, the RL agent period cost and action are the inventory period cost and replenishment order, respectively. For RSarsa, PRS, RSarsa.TD, and PRS.Beta, the extra information required by rumination is the inventory demand variable $\xi = d_t$.

The Q-value is implemented using grid tile coding [2] without hashing. Tile coding is a function approximation method based on a linear combination of weights of activated features, called "tiles." The approximation function with argument $\mathbf{z}$ is given by

$$f(\mathbf{z}) = w_1 \phi_1(\mathbf{z}) + w_2 \phi_2(\mathbf{z}) + \cdots + w_M \phi_M(\mathbf{z}), \quad (10)$$

where $w_1, w_2, \ldots, w_M$ are tile weights and $\phi_1(\mathbf{z}), \phi_2(\mathbf{z}), \ldots, \phi_M(\mathbf{z})$ are tile activation functions $\phi_i(\mathbf{z}) = 1$ only when $\mathbf{z}$ lies inside the hypercube of the ith tile.

The tile configuration, that is, $\phi_1(\mathbf{z}), \ldots, \phi_M(\mathbf{z})$, is predefined. Each Q-value is stored using tile coding through the weights. Given a value Q to store at any entry of $\mathbf{z}$, the weights are updated according to

$$w_i = w_i^{(\text{old})} + \frac{\left(Q - Q^{(\text{old})}\right)}{N}, \quad (11)$$

where $w_i^{(\text{old})}$ and $Q^{(\text{old})}$ are the weight (of the ith tile) and approximation before the new update. Variable N is for a number of tiling layers.

For P1, we use a tile coding with 10 tiling layers. Each layer has $8 \times 3 \times 4$ three-dimensional tiles, covering multidimensional state-action space of $[-300, 500] \times [0, 150] \times [0, 150]$ corresponding to $s = [x, b^{(1)}]$ and a. This means that this tile coding allows only a state lying in $[-300, 500] \times [0, 150]$ and a value of action between 0 and 150. The dimensions, along x, $b^{(1)}$, and a, are partitioned into 8, 3, and 4 partitions, creating 96 three-dimensional hypercubes for each tiling layer. All layers are overlapping to constitute an entire tile coding set. Layer overlapping is arranged randomly. For P2, we use a tile coding with 5 tiling layers. Each tiling has 11×5 two-dimensional tiles, covering the space of $[-300, 650] \times [0, 150]$ corresponding to $s = (x + b^{(1)})$ and a. For P3, we use a tile coding with 10 tiling layers. Each tiling has $8 \times 3 \times 3 \times 4$ four-dimensional tiles, covering the space of $[-400, 1200] \times [0, 150] \times [0, 150] \times [0, 150]$ corresponding to $s = [x, b^{(1)}, b^{(2)}]$ and a.

All RL agents use the ϵ-greedy policy with $\epsilon = 0.2$. The learning update uses the learning rate $\alpha = 0.7$ and discount factor $\gamma = 0.8$.

Figures 4, 5, and 6 show moving averages (of degree 1000) of period costs, in P1, P2, and P3, obtained with different learning agents, as indicated in the legends ("R.TD" is short for RSarsa.TD). Figures 7 and 8 show box plots of average costs obtained with the different methods in early and later periods, respectively.

The results are summarized in Table 1. The computation costs of the methods are measured by relative average computation time per epoch, shown in lines 1–3. Average

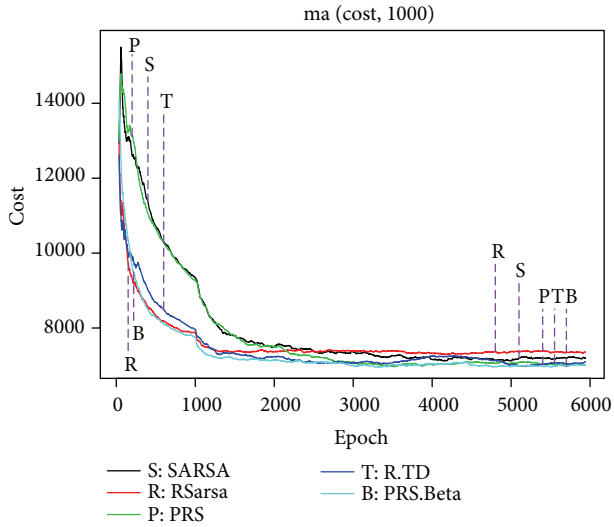

Figure 4: Moving average of period costs, P1.

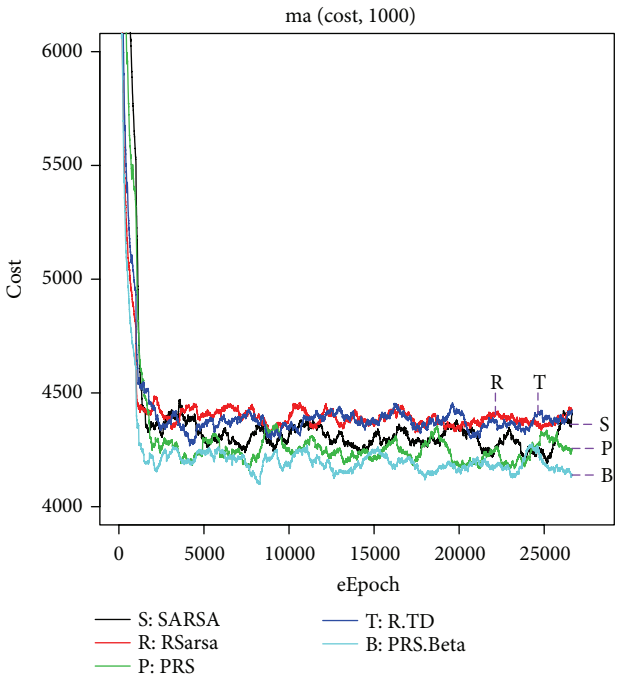

Figure 5: Moving average of period costs, P2.

costs are used as the inventory management performance and they are shown in lines 4–6 for early periods (periods 1–2000 in P1 and P2 and periods 1–4000 in P3) and lines 7–9 for later periods (periods after early periods). The numbers in each entry indicate average costs obtained from the corresponding methods. Parentheses reveal results from one-side Wilcoxon's rank sum tests: "W" indicates that the average cost is significantly lower than an average cost obtained from SARSA ($P < 0.05$); otherwise, the P value is shown instead.

The computation costs of RSarsa, PRS, and PRS.Beta (full rumination) are about 20–30 times of SARSA (RL without

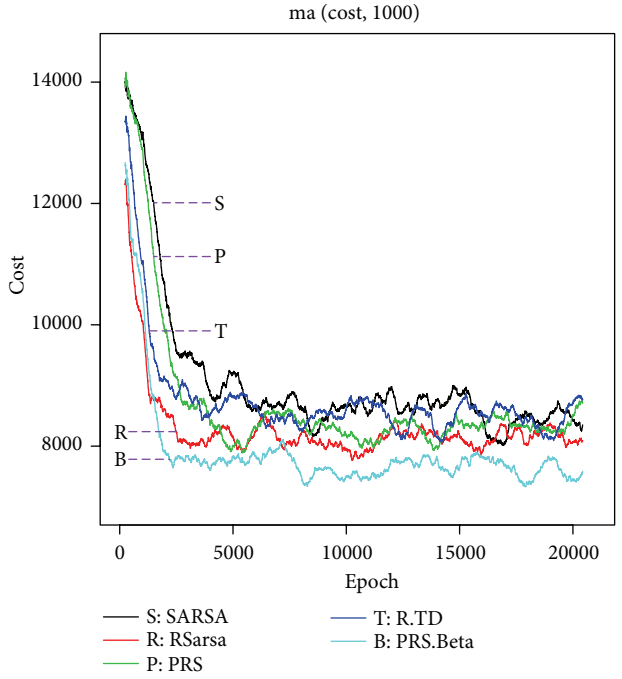

Figure 6: Moving average of period costs, P3.

rumination). RSarsa.TD (selective rumination) dramatically reduces the computation cost of rumination at scales of 5–7 times. An evaluation of the effectiveness of each method (compared to SARSA) shows that RSarsa and PRS.Beta significantly outperform SARSA in early periods for all 3 problems. Average costs obtained from RSarsa.TD are lower than ones from SARSA, but significance tests can confirm only results in P1 and P2. It should be noted that PRS results do not show significant improvement over SARSA. This agrees with results in a previous study [17]. With respect to performance in later periods, average costs of PRS and PRS.Beta are lower than SARSA's in all 3 problems. However, significance tests can confirm only few results (P1 for PRS and P1 and P2 for PRS.Beta).

Table 2 shows a summary of results from significance tests comparing the previous study's RRL methods (RSarsa and PRS) to our proposed methods (RSarsa.TD and PRS.Beta). The entries with "W" indicate that our proposed method on the corresponding column significantly outperforms a previous method on the corresponding row ($P < 0.05$). Otherwise, the P value is indicated.

6. Conclusions and Discussion

Our results have shown that PRS.Beta achieves our goal, which is to address the slow learning rate of PRS, as it significantly outperforms PRS in early periods in all 3 problems, and to address the long-term learning quality of RSarsa, as it significantly outperforms RSarsa in later periods in P1 and P2 and its average cost is lower than RSarsa's in P3. It should be noted that although the performance of RSarsa.TD may not seem impressive when compared to PRS.Beta's, RSarsa.TD

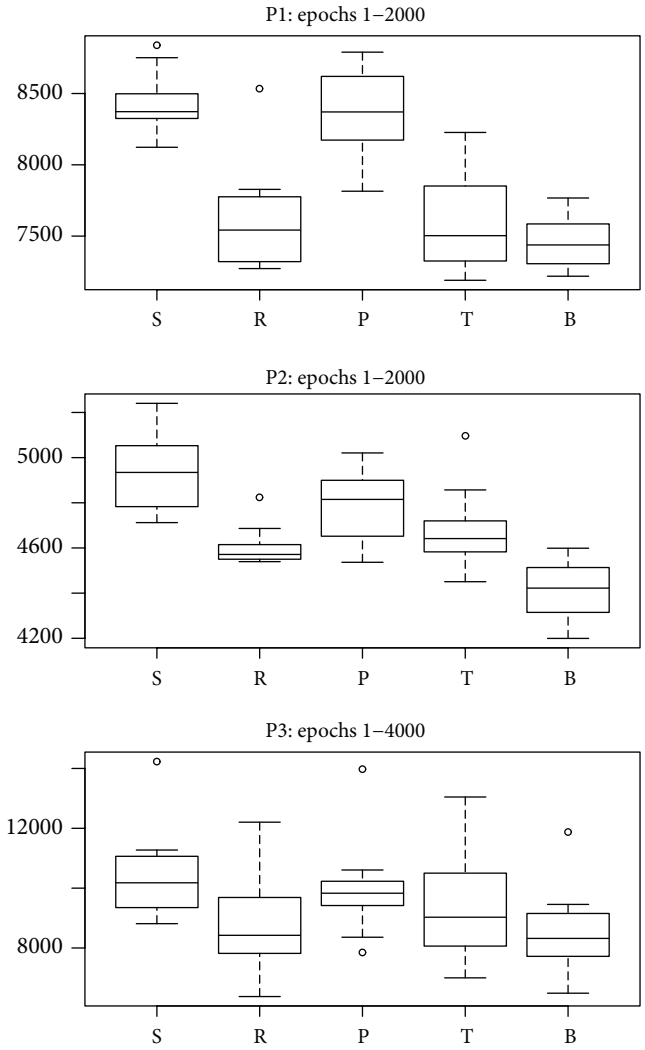

Figure 7: Average costs in early periods.

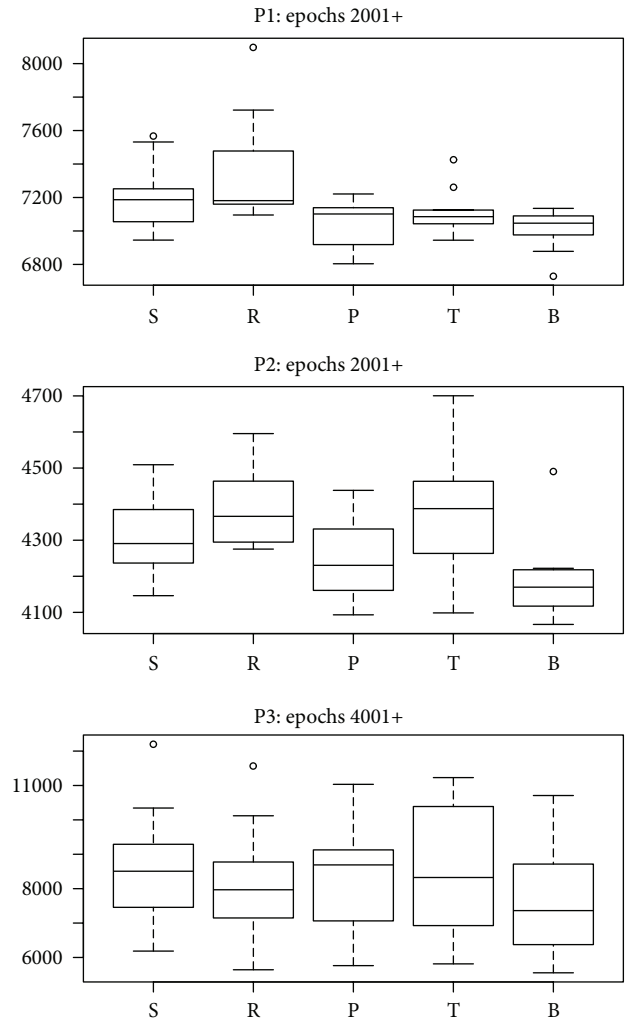

Figure 8: Average costs in late periods.

Table 2: Experimental results.

Line			RSarsa.TD	PRS.Beta
		Early periods		
1	P1	RSarsa	0.49	0.16
2		PRS	W	W
3	P2	RSarsa	0.95	W
4		PRS	0.10	W
5	P3	RSarsa	0.80	0.37
6		PRS	0.26	W
		Later periods		
7	P1	RSarsa	W	W
8		PRS	0.63	0.14
9	P2	RSarsa	0.46	W
10		PRS	0.97	0.12
11	P3	RSarsa	0.66	0.26
12		PRS	0.60	0.18

requires less computational cost. Therefore, as RSarsa.TD shows some improvement over SARSA, this reveals that selective rumination is still worth further study.

It should be noted that PRS.Beta employs TD error to control its behavior (6). The notion to extend TD error to determine learning factors is not limited only to rumination. It may be beneficial to use the TD error signal to determine other learning factors, such as the learning rate, for an adaptive-learning-rate agent. A high TD error indicates that the agent has a lot to learn, that what it has learned is wrong, or that things are changing. For each of these cases, the goal is to make the agent learn more quickly. So, a high TD error should be a clue to increase the learning rate, increase the degree of rumination, or increase the chance to do more exploration.

To address issues in RL worth investigation, more efficient Q-value representations should be among the priorities. Regardless of the action policy, every RL policy relies on Q-values to determine the action to take. Function approximations suitable to represent Q-values should facilitate efficient realization of an action policy. For example, ϵ-greedy policy has to search for an optimal action. A Q-value representation

suitable for an ϵ-greedy policy should allow efficient search for an optimal action given a state. Another general RL action policy is the *softmax* policy [2]. Given a state, the softmax policy has to evaluate the probabilities of candidate actions based on their associated Q-values. A representation that facilitates efficient mapping from Q-values to the probabilities would have great practical importance in this case. Due to the interaction between the Q-value representation and the action policy, there are considerable efforts to combine these two concepts. This is an active research direction under the rubric of *policy gradient RL* [18].

There are many issues in RL needed to be explored, theoretically and for application. Our findings reported in this paper provide another step in understanding and applying RL to practical inventory management problems. Even though we only investigated inventory management problems here, our methods can be applied beyond this specific domain. This early step in the study of using TD error to control learning factors, along with investigation of other issues in RL, would yield a more robust learning agent that is useful in a wide range of practical applications.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

The authors would like to thank the Colorado State University Libraries Open Access Research and Scholarship Fund for supporting the publication of this paper.

References

[1] D. Bertsimas and A. Thiele, "A robust optimization approach to inventory theory," *Operations Research*, vol. 54, no. 1, pp. 150–168, 2006.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning*, MIT Press, Boston, Mass, USA, 1998.

[3] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, Wiley Series in Probability and Statistics, Wiley-Interscience, Hoboken, NJ, USA, 2007.

[4] J. Rao, X. Bu, C. Z. Xu, L. Wang, and G. Yin, "VCONF: a reinforcement learning approach to virtual machines auto-configuration," in *Proceedings of the 6th International Conference on Autonomic Computing (ICAC '09)*, pp. 137–146, ACM, Barcelona, Spain, June 2009.

[5] S. G. Khan, G. Herrmann, F. L. Lewis, T. Pipe, and C. Melhuish, "Reinforcement learning and optimal adaptive control: an overview and implementation examples," *Annual Reviews in Control*, vol. 36, no. 1, pp. 42–59, 2012.

[6] A. Coates, P. Abbeel, and A. Y. Ng, "Apprenticeship learning for helicopter control," *Communications of the ACM*, vol. 52, no. 7, pp. 97–105, 2009.

[7] C. W. Anderson, D. Hittle, M. Kretchmar, and P. Young, "Robust reinforcement learning for heating, ventilation, and air conditioning control of buildings," in *Handbook of Learning and Approximate Dynamic Programming*, J. Si, A. Barto, W. Powell, and D. Wunsch, Eds., pp. 517–534, John Wiley & Sons, New York, NY, USA, 2004.

[8] R. Lincoln, S. Galloway, B. Stephen, and G. Burt, "Comparing policy gradient and value function based reinforcement learning methods in simulated electrical power trade," *IEEE Transactions on Power Systems*, vol. 27, no. 1, pp. 373–380, 2012.

[9] Z. Tan, C. Quek, and P. Y. K. Cheng, "Stock trading with cycles: a financial application of ANFIS and reinforcement learning," *Expert Systems with Applications*, vol. 38, no. 5, pp. 4741–4755, 2011.

[10] A. Castelletti, F. Pianosi, and M. Restelli, "A multiobjective reinforcement learning approach to water resources systems operation: pareto frontier approximation in a single run," *Water Resources Research*, vol. 49, no. 6, pp. 3476–3486, 2013.

[11] T. Katanyukul, E. K. P. Chong, and W. S. Duff, "Intelligent inventory control: is bootstrapping worth implementing?" in *Intelligent Information Processing VI*, vol. 385 of *IFIP Advances in Information and Communication Technology*, pp. 58–67, Springer, New York, NY, USA, 2012.

[12] T. Akiyama, H. Hachiya, and M. Sugiyama, "Efficient exploration through active learning for value function approximation in reinforcement learning," *Neural Networks*, vol. 23, no. 5, pp. 639–648, 2010.

[13] K. Doya, "Metalearning and neuromodulation," *Neural Networks*, vol. 15, no. 4–6, pp. 495–506, 2002.

[14] P. Dayan and N. D. Daw, "Decision theory, reinforcement learning, and the brain," *Cognitive, Affective and Behavioral Neuroscience*, vol. 8, no. 4, pp. 429–453, 2008.

[15] T. Katanyukul, W. S. Duff, and E. K. P. Chong, "Approximate dynamic programming for an inventory problem: empirical comparison," *Computers and Industrial Engineering*, vol. 60, no. 4, pp. 719–743, 2011.

[16] C. O. Kim, I. H. Kwon, and J. G. Baek, "Asynchronous action-reward learning for nonstationary serial supply chain inventory control," *Applied Intelligence*, vol. 28, no. 1, pp. 1–16, 2008.

[17] T. Katanyukul, "Ruminative reinforcement learning: improve intelligent inventory control by ruminating on the past," in *Proceedings of the 5th International Conference on Computer Science and Information Technology (ICCSIT '13)*, Academic Publisher, Paris, France, 2013.

[18] N. A. Vien, H. Yu, and T. Chung, "Hessian matrix distribution for Bayesian policy gradient reinforcement learning," *Information Sciences*, vol. 181, no. 9, pp. 1671–1685, 2011.