Ruminative Reinforcement Learning: Improve Intelligent Inventory Control by Ruminating on the Past

Tatpong Katanyukul
Khon Kaen University/Computer Engineering, Khon Kaen, Thailand
Email: tatpong @kku.ac.th

***Abstract*—Reinforcement Learning (RL) can solve practical sequential decision problems, even when structures of the problems are less understood. However, some sequential decision problems intrinsically have structural parts that are easily to formulate and distinguish from less understood parts. Exploiting this knowledge may help improve performance of RL. This study proposed and investigated an approach to exploit the knowledge of structural parts of inventory management problems in the context of RL. The proposed method is motivated by human behavior of ruminating on what has happened and what would happen if alternative choices would have been taken. Our investigation provides an insight into RL mechanism and our experimental results show viability of the approach.**

***Index Terms*—Temporal difference learning, ruminative behavior, markov decision problem, artificial intelligence, reinforcement learning, inventory control, approximate dynamic programming**

I. Introduction

Reinforcement Learning (RL) is an approach to solve practical sequential decision or sophisticated control problems, even when structures of the problems are less understood. RL has been studied extensively and applied in wide range of applications, including virtual machine configuration [1], robotics [2], helicopter control [3], ventilation, heating and air conditioning control [4], financial management [5], water resource management [6], and inventory management [7]. Prevalence of RL in current research is credited to RL's effectiveness, potential possibilities [8], link to mammal learning processes [9], and its model-free property [10].

Despite fascination of RL's model-free property, some application domain can naturally be formulated into a well-structured part interacting with a part that is less understood. An example is a domain of inventory management problems, costs of issuing replenishment order, handling inventory, and loss due to inventory shortage can be determined precisely in advance. On the other hand, customer demand, delivery time, availability of supplies or in an extreme case the length of the horizon is less predictable. However, once values of less predictable variables are known, the period cost can be precisely determined. It is as a warehouse would know its inventory cost after it has received its replenishment and seen the demand. The formulation to calculate period cost is a well-structured part, while another part, e.g., demand, is unpredictable. Knowledge about a well-structured part can be exploited, while learning mechanism takes care of the less understood part.

Inventory management is an essential business activity. Inventory problems appear in various forms and their forms often change over time. Many articles, including [11], addressed the need for an efficient and flexible inventory solution that is also simple to implement in practice. This may explain extensive studies of RL application to inventory management. Most of the previous works applying RL to inventory management mainly focus on learning-based schemes. However, there are some studies, e.g., [10] and [12], investigating simulation-based schemes. Nonetheless, learning- and simulation-based schemes are not mutually exclusive. Kim et al. [13] proposed asynchronous action-reward learning method. They used simulation to evaluate consequences of actions not taken in order to accelerate a learning process in a stateless system. An extension to a state-based system would allow applicability to a wider range of problems.

Our proposed method, "Ruminative Reinforcement Learning" (RRL), can be seen as this extension. The findings here provide an insight into RL mechanism. Our results show viability of RRL in different phases of operation. Regarding to applicability of the framework, another example is an area of water resource management. Precipitation, evaporation, and water demand are hard to predict. The capacities of reservoirs, size of a serving area, and length of irrigation canal are known. Once values of precipitation, evaporation, and demand are realized, how well the management can keep water levels as promised can be evaluated in high accuracy. It can be formulated in a similar manner. In additions, this framework can be applied to any sequential decision problem, where state and period cost variables can be determined precisely, once a stochastic variable is realized, as illustrated in Figure 2.

II. Background

Manuscript received July 25, 2013; revised August 20, 2013; accepted September 15, 2013.

doi:10.4304/jcp.9.7.1530-1535

The sequential decision problem can be formulated as a Markov Decision Problem (MDP) [14]. Given an action is determined by a policy π, the long-term state cost can be written as,

$$C^{\pi}(s) = r(s) + \gamma \Sigma_{s'} p^{\pi}(s'|s)\, C^{\pi}(s'), \quad (1)$$

where $r(s)$ is an expected state cost, γ is a discounted factor, $p^{\pi}(s'|s)$ is a probability that a state in the next step is s' given a state of the current step s. In practice, the expected long-term cost is difficult to find. RL provides a framework to approximate it and find a good action policy. An approximate long-term cost $Q(s) = r(s) + \gamma Q(s')$.

Temporal Difference (TD) learning uses TD error ψ (Eq. 2) to estimate the long-term cost in an iterative manner (Eq. 3),

$$\psi = r + \gamma Q(s',a') - Q(s,a), \quad (2)$$

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

where r is a sampling period cost, corresponding to take action a in state s; α is a learning rate; s' and a' are a state and an action taken in the next period, respectively.

Once values of $Q(s,a)$ are learned sufficiently, they are presumed to be good approximation of the long-term costs. SARSA [15], a widely used RL algorithm, uses Q(s,a)—often called "Q-value"— along with an action policy to determine an action to take. Q-value is updated based on TD learning (Eq. 2 and 3). In each period, an action is determined based on a policy π. An action policy π can be stochastic, such that it defines a probability to take an action a given state s: $\pi(s,a) = p(a|s)$. The policy has to balance between taking the best actions and try other actions, so that it can learn new possibilities. This is the issue of balancing exploitation and exploration, as discussed in [15].

An ε-greedy policy is a simple policy. At each period, with probability ε, the policy takes an action randomly picked from $a \in A(s)$, where $A(s)$ is a set of allowable actions given state s. Otherwise, it takes an optimal action a^* based on the current Q-value, $a^* = argmin_a Q(s,a)$.

III. Ruminative Reinforcement Learning

SARSA algorithm assumes that the agent knows only current state s, action it takes a, period cost r, next state s', and action it will take a'. Interaction of SARSA agent and its environment are shown in Fig. 1.

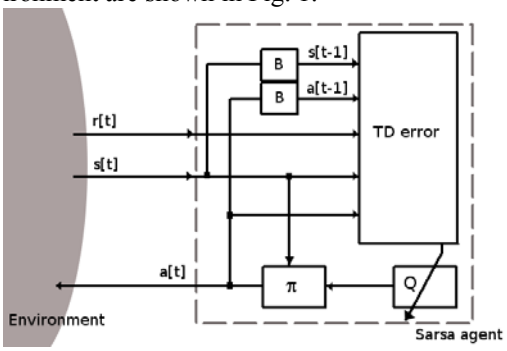

Figure 1. SARSA agent and interacting signals.

Each period, once period cost r and next state s' are realized, SARSA algorithm updates Q-value based on TD error (Eq. 2 and 3), requiring only values of s, a, r, s', and a'. However, in some problem domains, we may have extra knowledge about the problem and want to take this advantage.

Suppose the problem structure can naturally be formulated such that period cost r and next state s' are determined by a function k: s, a, $\xi \mid\!\rightarrow r$, s', where ξ is an extra information. Variable ξ is assumed to capture the stochastic aspect of the problem. The process generating ξ is unknown, but the value of ξ is fully observable, after the period is over. Given a value of ξ, along with s and a, deterministic function k can determine r and s', precisely.

Original SARSA updates only one entry of $Q(s,a)$ each period, since it can provide only one set of required values of (s, a, r, s', a'). With mapping function k and an observed value of ξ, we can do rumination: evaluating consequences of other actions $\hat{a}$'s that were not taken.

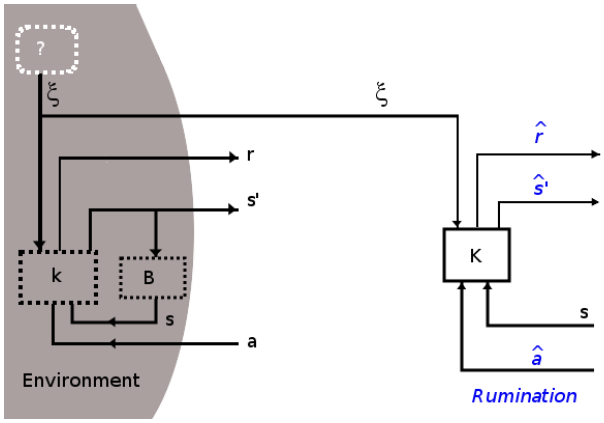

Figure 2. Knowledge of problem's structure and rumination.

TABLE I.

RSARSA: Reinforcement Learning with Rumination

Initialize $Q(s,a)$.
Observe s.
Determine a by policy π.

For each period
  Observe r, s', and ξ.
  Determine a' by policy π.
  $\psi := r + \gamma Q(s',a') - Q(s,a)$.
  $Q(s,a) := Q(s,a) + \alpha \cdot \psi$.
  For each $\hat{a} \in \hat{A}$(s)
  (1) calculate □, $\hat{s}'$ by k(s, $\hat{a}$, ξ),
  (2) determine $\hat{a}'$,
  (3) calculate $\psi := \square + \gamma Q(\hat{s}', \hat{a}') - Q(s, \hat{a})$,
  (4) update $Q(s, \hat{a}) := Q(s, \hat{a}) + \alpha \cdot \psi$.
  Until ruminating all $\hat{a} \in \hat{A}$
  s := s'; a := a'.
Until termination

Rumination can be done by (1) choosing a ruminative action $\hat{a} \in \hat{A}(s)$ and (2) use k(s, $\hat{a}$, ξ) to determine ruminative period cost □ and next state $\hat{s}'$. (3) Then, ruminative next action $\hat{a}'$ can be determined with policy π. (4) Values of (s, $\hat{a}$, □, $\hat{s}'$, $\hat{a}'$) can be used to evaluate Eq. 2 and 3. Table 1 shows Ruminative SARSA (RSARSA) algorithm.

Fig. 2 illustrates knowledge of problem's structure (on the left side) and our proposed rumination (on the right) resembling the problem's structure to provide □ and $\hat{s}'$.

As our experimental results shown in Section 4, RSARSA has shown significant improvement over SARSA performance in early periods. However, its performance in later periods was shown to be inferior to SARSA.

This may be explained by that, despite having faster convergence rate, RSARSA's approximate long-term costs may converge to wrong values. Lack of transition probability $p^{\pi}(s'|s)$, Long-term cost estimated by TD learning (Eq. 2 and 3) relies on sampling trajectory: natural state-action visitation provides frequency of updates, equivalent to $p^{\pi}(s'|s)$.

However, rumination mechanism employed in RSARSA does not take this visiting frequency into account. Since rumination is done over an action set, it does not spoil state visiting frequency. To correct, action visiting frequency, probability of visiting each action should be accounted.

Given state s and ε-greedy policy, probability of taking action $\hat{a}$:

$$p(\hat{a}) = \varepsilon/|\hat{A}(\mathrm{s})|, \text{ for } \hat{a} \neq \hat{a}^* \quad \text{and}$$

$$p(\hat{a}) = \varepsilon/|\hat{A}(\mathrm{s})| + (1-\varepsilon), \text{ for } \hat{a} = \hat{a}^*, \qquad (4)$$

where $|\hat{A}(\mathrm{s})|$ is a size of a ruminative action set and $\hat{a}^* = \mathit{argmin}_{\hat{a}}\ Q(s,\hat{a})$. An algorithm for Policy-weighted RSARSA (PRS) is shown in Table 2.

TABLE II
PRS: POLICY-WEIGHTED RSARSA

Initialize $Q(s,a)$.
Observe s.
Determine a by policy π.

For each period
Observe r, s', and ξ.
Determine a' by policy π.
$\psi := r + \gamma Q(s',a') - Q(s,a)$.
$Q(s,a) := Q(s,a) + \alpha \cdot \psi$.
For each $\hat{a} \in \hat{A}(\mathrm{s})$,
(1) calculate □, $\hat{s}'$ by $k(\mathrm{s}, \hat{a}, \xi)$,
(2) determine $\hat{a}'$,
(3) calculate $\psi := \square + \gamma Q(\hat{s}', \hat{a}') - Q(s, \hat{a})$,
(4) calculate $p(\hat{a})$ according to policy π (Eq. 4 for ε-greedy),
(5) $\beta = \alpha \cdot p(\hat{a})$,
(6) update $Q(s, \hat{a}) := Q(s, \hat{a}) + \beta \cdot \psi$.
Until ruminating all $\hat{a} \in \hat{A}$
$s := s'$; $a := a'$.
Until termination

The computational cost increased by rumination varies directly to the size of ruminative action set: that is roughly $|\hat{A}(s)|$ times of SARSA. Although this may not be critical in inventory management, it is still worth to investigate an opportunity to improve rumination on this aspect. We have tried to use TD error ψ to control when to do rumination. This is based on intuition that high magnitude of TD error associates to an agent's urge to learn fast and the urge declines as TD error fades: when an agent is put in a new environment; the environment is changing; or, what an agent knew is not correct; any of these situations would cause high TD error.

In this early stage of an attempt using TD error to guide rumination, each period we use magnitude of a normalized TD error to define the probability to do rumination:

$$p(\text{rumination}) = 1 - exp(-|2\ \psi/(r + Q(s,a)|) . \qquad (5)$$

In Eq. 5, value of probability is in [0,1). Large magnitude ψ gives a large probability to do rumination, and vice versa. For example, if ψ = average of r and $Q(s,a)$, the chance to do rumination is 62%.

IV. EXPERIMENTS AND RESULTS

Our study uses computer simulations to conduct numerical experiments, on two inventory management problems. Both problems are periodic review single-echelon with one-period leadtime and nonzero setup cost. The same markov model are used to govern both problem environments, but with different settings. The problem state space is $\boldsymbol{I} \times \{0, \boldsymbol{I}^+\}$, for on-hand and in-transit inventories: x and b, respectively. The action space is $\{0, \boldsymbol{I}^+\}$, for replenishment order a. State transitions are specified by (1) $x_{t+1} = x_t + b_t - d_t$ and (2) $b_{t+1} = a_t$, where d_t is normally distributed. The inventory period cost is calculated from $r_t = K\ \delta(a_t) + G\ a_t + H\ a_t\ \delta(a_t) - B\ a_t\ \delta(-a_t)$, where K, G, H, and B are setup, unit, holding, penalty costs, respectively; and $\delta(\cdot)$ is a step function.

Four RL agents, SARSA, RSARSA, PRS, and PRS.TD, are tested. PRS.TD is an abbreviation for PRS algorithm with rumination controlled by TD error, as described in Eq. 5. Each experiment is repeated for 10 times. In each repetition, an agent is initialized with all zero Q-values. Then, it is run consecutively for N_E episodes. Each episode starts with random initial state and action and ends when the episode has reached N_P periods or an agent has visited a termination state, which are states lying outside a valid range of Q-value implementation. The maximum number of periods in each episode, N_P, defines length of a problem horizon, while the number of episodes N_E specifies variety of problem scenarios, i.e., different initial states and actions.

Two problem settings are used in our experiments. Problem 1 (P1) has $N_P = 1000$, $N_E = 200$, $K = 100$, $G = 100$, $H = 20$, and $B = 200$. Environment's state $[x,b]$ is set as RL agent's state: $s = [x,b]$. Therefore, RL agent's state is two-dimensional. Problem 2 (P2) has $N_P = 60$, $N_E = 500$, $K = 200$, $G = 50$, $H = 20$, and $B = 200$. RL agent's state is set as an inventory level: $s = x + b$. RL agent's state is one-dimensional.

For both problems, demand d_t is normally distributed with mean 50 and variance 100. RL agent's period cost and action are inventory period cost and replenishment order, respectively. For ruminative algorithms, the extra information is inventory's demand variable: $\xi = d_t$.

For all RL agents, Q-value is implemented with grid tile coding, as explained in [15], without hashing. Tile

coding is a function approximation, based on linear combination of weights of activated tiles. The suitability and justification of using tile coding for RL is discussed in [16]. Approximate function of variable z,

$$f(z) = w_1 \cdot \phi_1(z) + w_2 \cdot \phi_2(z) + . \, w_M \cdot \phi_M(z), \quad (5)$$

where $w_1, w_2, \ldots, w_M$ are tile weights, and $\phi_1(z)$, $\phi_2(z)$, …, $\phi_M(z)$ are tile activation function: $\phi_i(z) = 1$, only when z lies within the i^{th} tile hypercube.

Tile configuration, i.e., $\phi_1(z)$, $\phi_2(z)$, …, $\phi_M(z)$, are predefined. Q-value is stored in tile coding through weights. Given a value Q to store at an entry of z, weights can be updated with $w_i = Q/N$, for all i's of tiles activated by z, where N is a number of activated tiles. However, in our preliminary experiments, we found that it is more effective to use:

$$w_i = w_i^{(old)} + (Q - Q^{(old)})/N, \quad (6)$$

where $w_i^{(old)}$ and $Q^{(old)}$ are weight and approximation before new update. We use Eq. 6 to update tile coding in all our experiments.

For P1, we use tile coding with 5 tiling layers. Each layer has 11×5×3 three-dimensional tiles, covering space of [−300,500] × [0,150] × [0,150] corresponding to s=[x, b] and a. This means that this tile coding allows only a state lying in [−300,500] × [0,150] and a value of action between 0 and 150. Dimensions, along x, b, and a, are partitioned into 11, 5, and 3 partitions, creating 165 three-dimensional hypercubes for each tiling layer. All layers are overlapping to constitute an entire set of tile coding. Layer overlapping is arranged randomly. For P2, we use tile coding with 5 tilings. Each tiling has 11×5 two-dimensional tiles, covering space of [−300,650] × [0,150] corresponding to s=(x+b) and a.

All RL agents are set up with ε = 0.2 (for ε-greedy policy), learning rate α = 0.7, and discounted factor γ = 0.8.

The results are summarized in Table 3. PRS has shown to outperform SARSA in both problems: its average costs 6692 and 4288 of P1 and P2, respectively (shown on rows 6 and 7) are lower than SARSA's 6777 and 4355. Both cases are also confirmed by Wilcoxon rank sum test (WT) with significant level 0.05 (indicating, on rows 9 and 10, by 'W', meaning that its average cost is significantly lower than SARSA).

Regarding to RSARSA, its average costs are higher than SARSA in both problems. Each case is confirmed by WT (indicating by 'L', meaning that its average cost is significantly higher than SARSA).

Our attempt of TD-error-controlled rumination turns out with mixed results: PRS.TD outperformed SARSA in P1, but underperformed in P2.

Despite inferior overall performance, RSARSA outperformed SARSA in early periods (1-300) in both P1 and P2 (rows 12-15). PRS and PRS.TD also outperformed SARSA in P2 in these early periods. In P1, both PRS and PRS.TD has lower average costs than SARSA, but these cannot be confirmed by WT (p values 0.529 and 0.684 for PRS and PRS.TD, respectively). Fig. 3 illustrates average costs obtained in P1 (left plots) and P2 (right plots) during periods 1-300 (top plots), periods 301-3000 (middle row plots), and periods after 3000 (bottom plots). Top plots show that RSARSA also outperformed PRS and PRS.TD in early periods. However, for later periods, RSARSA underperformed others.

Fig. 4 shows moving average of period costs (left plots) and TD error ψ (right plots) of P1 (upper plots) and P2 (lower plots). Legends are shown in the top left plot. All four plots show that RSARSA has the fastest rate of convergence. However, it converged to suboptimal value, as revealed by the plots of TD error: RSARSA's ψ converges to higher value than the others in both P1 and P2. It should be noted that as the algorithm converges average cost declines and so does TD error. This is one of our motivations to use TD error as a clue to control rumination, in order to reduce the additional computational costs.

For each period, computation of RSARSA spent 10.59 and 20.28 times of SARSA in P1 and P2, respectively (rows 3-4). PRS spent a little bit longer than RSARSA. These computation times correspond to size of ruminative action space $|\hat{A}(s)|$, here as partitions of action space 15 and 25 in P1 and P2, respectively. Partitioning is nature of tile coding, implemented Q-value. PRS.TD spent only 2.19 and 3.12 times of SARSA.

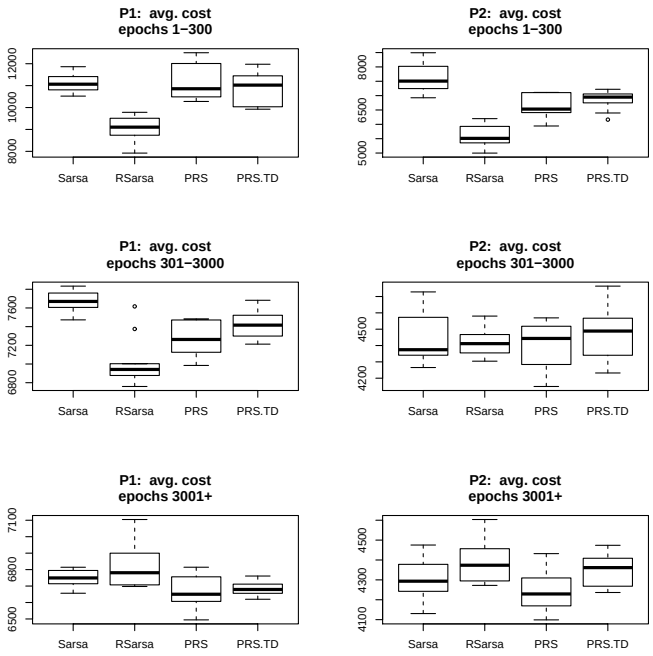

Figure 3. Average period costs at different period ranges: boxplot shows maximum, 3rd-quantile, median, 1st-quantile, minimum, and outliers of average costs obtained by SARSA, RSARSA, PRS, and PRS.TD.

V. Discussion and Conclusions

PRS has shown to be a viable algorithm, when structural part of the problem can be distinguished. Although RSARSA may converge to suboptimal policy in long run, it is shown to be a formidable candidate for short run. Therefore, it may be suitable for finite, especially short, horizon applications.

With major improvement of rumination's computing times, PRS.TD showed good results in early periods. However, its performance in long run reveals that attempt to control rumination by TD error requires further investigation. In addition, based on our intuition that high TD error associates to need for fast learning, the concept of PRS.TD can be extended to other aspects, such as using TD error to control learning rate or degree of exploration. RL has been associated to mammalian learning, as TD error is associated to dopamine activity. So are learning rate and degree of exploration to acetylcholine and noradrenaline, respectively [17]. The knowledge of interaction of these chemicals from neuroscience may reveal key mechanism to develop a self-adjusting RL agent, or vice versa.

Our results show domination of RSARSA in fast learning and PRS in quality of what it learns. In PRS, we use simple term $\beta = \alpha \cdot p(\hat{a})$ for ruminative learning rate, where α is the same learning rate, used in regular TD learning. The development of better ruminative learning rate may lead to an algorithm that is both fast learning and good in long run.

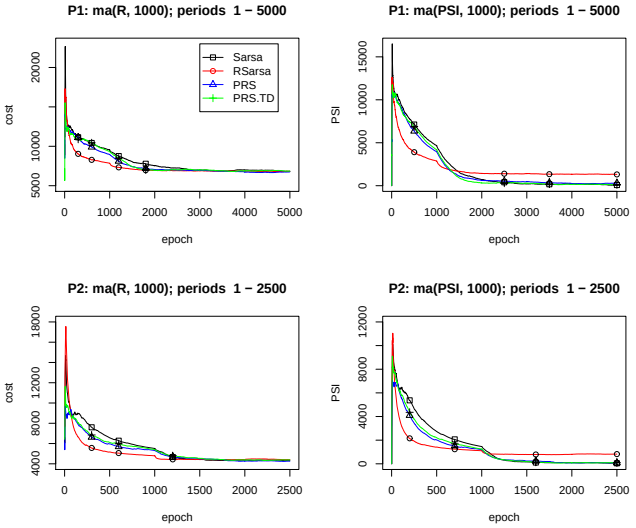

Figure 4. Moving average of period costs.

EXPERIMENTAL RESULTS

Row		SARSA	RSARSA	PRS	PRS.TD
Relative computation time/epoch					
3	P1	1	10.59	11.95	2.19
4	P2	1	20.28	20.91	3.12
Average cost (AC)					
6	P1	6777	6830	6692	6709
7	P2	4355	4406	4288	4392
Wilcoxon rank sum test (WT)					
9	P1		L	W	W
10	P2		L	W	L
Periods 1-300					
12	P1: AC	11144	9042	11129	10898
13	P1: WT		W	p 0.529	p 0.684
14	P2: AC	7604	5562	6621	6844
15	P2: WT		W	W	W

Lastly, every component of RL affects its overall performance, to develop more effective and efficient RL application, we may need to consider state-of-the-arts, such as policy gradient, transfer learning, active learning, adaptive step size, adaptive action search space, and efficient Q-value representation, including deep learning. Regarding efficient Q-value representation, widely used functions, i.e., look-up tables, neural networks, radial basis functions, and tile coding, are good for interpolation, but RL applications needs extrapolation, as well. If this need is met, it would yield more efficient RL applications as well as faster developing processes.

REFERENCES

[1] J. Rao, X. Bu, C.Z. Xu, L. Wang, and G. Yin, "VCONF: a reinforcement learning approach to virtual machines auto-configuration," in *Proc. 6th int. conf. autonomic computing*, Barcelona, Spain, ACM, 2009, pp. 137–146.

[2] S.G. Khan, G. Herrmann, F.L. Lewis, T. Pipe, and C. Melhuish, "Reinforcement learning and optimal adaptive control: An overview and implementation examples," *Annu. Rev. in Control*, vol. 36, 42–59, 2012.

[3] A. Coates, P. Abbeel, and A.Y. Ng, "Apprenticeship learning for helicopter control," *Commun. ACM*, vol. 52, 2009.

[4] 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*, Si, Barto, Powell, and Wunsch Eds. John Wiley & Sons, 2004.

[5] Z. Tan, C. Quek, and P.Y.K. Cheng, "Stock trading with cycles: a financial application of ANFIS and reinforcement learning," *Expert Syst. Appl.*, vol. 38, pp. 4741–4755, 2011.

[6] 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 Resour. Res.*, 2013.

[7] T. Katanyukul, E.K.P. Chong, and W.S. Duff, "Intelligent inventory control: is bootstrapping worth implementing?" in *Intelligent Information Processing VI*, Springer, Vancouver, BC., 2012.

[8] T. Akiyama, H. Hachiya, and M. Sugiyama, "Efficient exploration through active learning for value function approximation in reinforcement learning," *Neural Networks*, vol. 23, pp. 639–648, 2010.

[9] A.M. Bornstein and N.D. Daw, "Multiplicity of control in the basal ganglia: Computational roles of striatal subregions," *Curr. Opin. Neurobiol.*, vol. 21(3), pp. 374–380, 2011.

[10] T. Katanyukul, W.S. Duff, and E.K.P. Chong, "Approximate dynamic programming for an inventory problem: Empirical comparison," *Comput. Ind. Eng.*, vol. 60, pp.719–743, 2011.

[11] D. Bertsimas and A. Thiele, "A robust optimization approach to inventory theory," *Oper. Res.*, vol. 54(1), pp. 150–168, 2006.

[12] J. Choi, M.J. Realff, and J.H. Lee, "Approximate dynamic programming: application to process supply chain management," *AIChe J.*, vol. 52(7), pp. 2473–2485, 2006.

[13] C.O. Kim, I.H. Kwon, and J.G. Baek, "Asynchronous action-reward learning for nonstationary serial supply chain inventory control," *Appl. Intell.*, vol. 28(1), pp. 1–16, 2008.

[14] W.B. Powell, *Approximate Dynamic Programming*, John Wiley & Sons, 2011.

[15] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.
[16] S. Whiteson, M.E. Taylor, and P. Stone, "Adaptive tile coding for value function approximation," *AI Technical Report AI-TR-07-339*, University of Texas at Austin, 2007.
[17] K. Doya, "Metalearning and neuromodulation," *Neural Networks*, vol. 15, 2002.

Tatpong Katanyukul, born in Khon Kaen, Thailand June 1974, graduated B.Eng (electronics engineering) and M.Eng (computer science) from King Mongkut's Institute of Technology Ladkrabang and Asian Institute of Technology, respectively. Both institutions are in Bangkok, Thailand. He earned his Ph.D. (Mechanical Engineering), from Colorado State University, Colorado, USA in 2010.

Dr. Katanyukul currently works as a lecturer for department of computer engineering, Khon Kaen University, Thailand. His research interests are reinforcement learning and applications of other machine learning techniques.