Introduction of Fixed Mode States into Online Profit Sharing and Its Application to Waist Trajectory Generation of Biped Robot

Seiya Kuroda¹, Kazuteru Miyazaki², and Hiroaki Kobayashi³

¹ Panasonic Factory Solutions Co., Ltd.

² National Institution for Academic Degrees and University Evaluation,
1-29-1 Gakuennishimachi, Kodaira, Tokyo 187-8587, Japan, teru@niad.ac.jp
<http://svrrd2.niad.ac.jp/faculty/teru/index.html>

³ Meiji University, 1-1-1 Higashimita Tama-ku, Kawasaki, Kanagawa 214-8571, Japan

Abstract. In reinforcement learning of long-term tasks, learning efficiency may deteriorate when an agent's probabilistic actions cause too many mistakes before task learning reaches its goal. The new type of state we propose – *fixed mode* – to which a normal state shifts if it has already received sufficient reward – chooses an action based on a greedy strategy, eliminating randomness of action selection and increasing efficiency. We start by proposing the combining of an algorithm with penalty avoiding rational policy making and online profit sharing with fixed mode states. We then discuss the target system and learning-controller design. In simulation, the learning task involves stabilizing of biped walking by using the learning controller to modify a robot's waist trajectory. We then discuss simulation results and the effectiveness of our proposal.

Keywords: Reinforcement Learning, Exploitation-oriented Learning, Profit Sharing, Improved PARP, biped robot

1 Introduction

Robots working with human beings in a common living space must adapt autonomously to unknown environments based on sensory input and actions. *Reinforcement learning* (RL)[16], a type of such control, does not assume the existence of a priori knowledge about the environment, focusing instead on goal directed learning from interaction with the environment using rewards and a penalties as teaching signals.

In its focus on applications [14, 19, 6, 15, 8, 5, 7, 9, 18]. to control problems [14, 19, 6, 5, 9], RL is used for a wider variety of classes than a control-theory-based approach because it does not assume a priori knowledge about the controlled object. In the *value function* approach [14, 19], for example, state values or state-action values are estimated, and in the *direct policy search* approach [17, 6], control policy parameters are optimized. Hybrid approaches [5, 9] combined with RL and evolutionary computing such as *genetic algorithms* [4] reduce the number of trial-and-error searches.

These “royal road” approaches pursuing optimality, but for this they need much trial-and-error searching to ensure such optimality. This makes them difficult to apply to problems requiring fast decisions in long-term tasks.

We pursue rationality rather than optimality. We likewise seek to reduce the amount of trial-and-error searching by enhancing successful experience in the process termed *exploitation-oriented learning* (XoL) [13]. XoL learning approaches include the profit sharing (PS) [10] rationality theorem, penalty avoiding rational policy making (PARP) algorithm [11], and PS-r# [13].

We propose an algorithm that learns efficiently in long-term tasks. In applying previous learning methods to long-term tasks, we found that they failed before achieving their purpose due to the influence of randomness in their action selection. To avoid this, we focused on online PS [2] – a type of XoL proposed to obtain rewards efficiently online – and improved PARP) [18] – proposed to avoid penalties more efficiently in real-world applications rather than PARP.

Section 2 defines our problem domain. We treat both discrete and continuous sensory input directly and discretized by a *basis function* [12, 18]. Section 3 introduces a *fixed mode state* shifted from a normal state to online PS when it has enough rewards. We also present an overall algorithm to fit our task. Section 4 applies our proposal to a long-term continuous task generating a biped robot’s trajectory. Section 5 discusses simulation results. Section 6 frames conclusions.

2 Problem Domain

Consider an agent in an unknown environment. After receiving sensory input from the environment, the agent selects and executes an action. Time is discretized by a single input-action cycle. We assume that sensory input from the environment is the N_s dimensional vector of both continuous and discrete variables. We discretize continuous-discrete sensory input space into discrete state space $\mathcal{S} = \{s_k, k = 1, 2, \dots, N_B\}$ using a *basis function* [12, 18].

We also treat N_m dimensional action output in which individual components of output have N_a alternatives to be selected. We define them as follows:

- The set of alternatives of the i –th component, $i = 1, 2, \dots, N_m$ is given as $\mathcal{A}_i = \{a_{i,j} : j = 1, 2, \dots, N_a\}$.
- Action act_q is constructed as $act_q = (a_{1,j_q}, a_{2,j_q}, \dots, a_{N_m,j_q})$, in which $a_{i,j_q} \in \mathcal{A}_i, i = 1, 2, \dots, N_m$. The set of all actions therefore is $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \dots \times \mathcal{A}_{N_m}$ and the number of total actions is $|\mathcal{A}| = (N_a)^{N_m}$.

The pair of state s_k and action act_q selected in state (s_k, act_q) termed a *rule*. In PS and most RL, a parameter termed *weight* or *Q-value* is assigned to individual rules to evaluate their importance. A series of rules that begins from a reward/penalty state or an initial state and ends with the next reward/penalty state is termed an *episode*. If an episode contains rules of the same state but paired with different actions, the partial series from one state to the next is termed a *detour*. A rule always existing on a detour is termed an *irrational rule*, but otherwise a *rational rule*.

An agent gets a reward or a penalty based on a result of an action sequence. A reward is given when our purpose is achieved, so the reward reinforces the weights of all rules in the episode using an evaluation function. A penalty is given if our purpose fails. The rule directly responsible for the failure receives the penalty and is termed a *penalty rule*. If all selective rules for a state are penalty or irrational rules, the state is termed a *penalty state*. If a rule brings the current state to a penalty states, the rule is also classified as a penalty rule. A penalty rule is found using a *penalty rule decision* (PRD) procedure [12, 18].

We deal with the following penalties:

- There are $N_p - 1$ types of penalties p_ℓ ($\ell = 1, 2, \dots, N_p - 1$) in which p_1 has the highest priority that should be avoided, and p_{N_p-1} has the lowest priority.
- We also introduce additional penalty p_{N_p} to avoid a penalty state. Its priority is less than that of p_{N_p-1} .

Note that a penalty differs greatly from a negative reward and that our algorithm deals with the penalty and the reward independently of each other.

A function that maps states to actions is termed a *policy*. A policy with a positive amount of reward acquisition expectations is termed a *rational policy*, and a rational policy receiving no penalty is termed a *penalty avoiding rational policy*. Using PRD, we get a penalty avoiding rational policy.

3 Introduction of Fixed Mode States into Online PS

3.1 Profit Sharing

Profit Sharing (PS) reinforces all of the weights of rules on an episode at the same time as an agent gets a reward. It must store the episode $\{(s_1, act_1), \dots, (s_i, act_i), \dots, (s_T, act_T)\}$ in memory, where s_1 is an initial state and s_T is a reward-giving state. The number of rules in an episode is termed an *episode length*. If the agent selects action act_T at state s_T and gets reward r , the weight of each rule in the episode is updated as $w(s_i, act_i) \leftarrow w(s_i, act_i) + f(i)$ where $f(i)$ is termed the evaluation function. In general, geometrically decreasing function $f(i) = r\gamma^{T-i}$ is used where γ is termed a *discounted rate* with a range of $\gamma \in (0, 1]$. The rationality theorem about γ has been derived in [10]. The geometrically decreasing function satisfies the theorem. In simulation, we set $\gamma = 0.8$.

3.2 Online PS

PS requires much memories (storage) to save all episodes until an agent receives a reward. Computational complexity required to update weights depends on episode length T because all rules in the episode are updated at once when the agent gets a reward. Online PS proposed in [2] solves these problems using the two ideas detailed below.

Solution for Memory (Storage) Requirement

If we use the geometrically decreasing function, $f(i)$ approaches 0.0 exponentially, implying that it is not necessary to save rules for which $f(i)$ values are close to 0.0. Online PS then needs only to store in memory those rules of a certain length before a reward-giving time and must update only the weights of these rules. We term the length a *queue*. If the episode length exceeds the length of the queue, the rule stored in memory on the first is discarded, so the memory (storage) requirement is bounded by queue length.

Solution for Computational Complexity

For the rule in the queue, $w(s, act) \leftarrow w(s, act) + rc(s, act)$ and $c(s, act) \leftarrow \gamma c(s, act)$ have been executed every step where $c(s, act)$ is termed the *eligibility trace*, and is used in much RL [16, 3]. At the start of our learning, all $c(s, act)$ s for any state and action are initialized at 0.0 and the queue is emptied. When rule (s, act) is selected and put in the queue, 1.0 is added to $c(s, act)$ of the rule. $c(s, act)$ is updated when the rule is selected and decreases geometrically if the rule is not selected. If the agent selects a rule selected once and already stored in memory in the queue, the previous rule is discarded to avoid updating the same rule twice, so computational complexity is again bounded by queue length.

3.3 Fixed Mode State on Online PS for Long-Term Task

One weak point of PS with the geometrically decreasing evaluation function is learning inefficiency in a state far from a reward-giving state because the contribution of the reward to weight through the function becomes very small.

We therefore used two strategies – learning scheduling and penalty implementation. In learning scheduling, the PS learning task is broken down to a series of subtasks, from shortest term tasks to full term tasks. With penalty implementation, we expect that the number of candidate actions in early stage states will decrease rapidly and learning be speeded up.

Even so, for a very long-term task, the random selection strategy of actions makes PS learning very difficult. We therefore introduce a *fixed mode state*, meaning that we regard the state in which the agent has learned enough as the fixed mode state. In fixed mode, the weight is fixed and the action of the highest weight value is selected deterministically – *greedy selection*.

We consider two strategies regarding a normal state as a fixed mode state:

- **Strategy 1:** If the state gets a reward K times, the state is regarded as fixed mode.
- **Strategy 2:** When one subtask is finished, every state getting a reward in the task is regarded as fixed mode.

If the agent gets a penalty, up to three fixed mode states ahead the state are also returned to non fixed mode states to relearn around these states.

3.4 Rule Decomposition

As stated in Section 2, action space has $(N_a)^{N_m}$ elements, meaning that the agent must choose one action from $(N_a)^{N_m}$ actions for each state and needs three $(N_B \times (N_a)^{N_m})$ -dimensional weight matrices, $\mathbf{Q}$, $\mathbf{P}$, and $\mathbf{E}$, as shown later. If we have N_B discrete states, there are $N_B \times (N_a)^{N_m}$ rules. To reduce the number, we break rule (s_k, act_q) down into a set of sub-rules $\{(s_k, a_1 = a_{1,j_1}), (s_k, a_2 = a_{2,j_2}), \dots, (s_k, a_{N_m} = a_{N_m,j_{N_m}})\}$, where $a_i \in \mathcal{A}_i$. We choose one value from $\mathcal{A}_i$ for each a_i independently of other components and these sub-rules are simultaneously given a reward or a penalty. In this new framework, the agent needs only three $(N_B \times N_M \times N_a)$ -dimensional matrices.

We find a penalty rule efficiently through this new framework, but there remains the possibility of judging a penalty rule that is not a penalty rule as a penalty rule. We introduce the N_0 to avoid the mistake. We execute the judgment of a penalty rule for the rule whose number of times in selection exceeds or equals N_0 . For this new framework, penalty threshold λ is introduced and a subrule is classified as a penalty subrule if the failure rate exceeds λ . In the simulation, we set $\lambda = 0.5$. In simulation, we set $N_0 = 7$ based on preliminary experiments.

3.5 Action Selection in Online-PS

The agent selects an action in discrete state s_k as follows:

- If state s_k is neither a penalty state nor a fixed mode state, the agent selects an action from actions not in penalty rules based on the probability distribution proportional to weight $Q_k[i][j]$.
- If state s_k is a fixed mode state, the agent selects the action in the rule that has the largest weight $Q_k[i][j]$.
- If state s_k is a penalty state, the agent avoids a penalty by stochastically accounting for the priority of a penalty. First, we define $\bar{\mathcal{A}}_i(\ell)$ as a set of actions whose priority is lower than or equal to p_ℓ . $\bar{\mathcal{A}}_i(3)$, for example, is a set of actions not included in penalty rules for p_1, p_2 , and p_3 . If $\bar{\mathcal{A}}_i(N_p) = \phi$, the agent selects an action randomly from $\mathcal{A}_i$. If $\bar{\mathcal{A}}_i(\ell) \neq \phi$ but $\bar{\mathcal{A}}_i(\ell+1) = \phi$, then the agent selects an action in $\bar{\mathcal{A}}_i(\ell)$ based on *ranking selection* with probability $P_r(a_i = a_{i,k_i}) = \frac{2(\bar{N}_{a_i} - k_i + 1)}{\bar{N}_{a_i}(\bar{N}_{a_i} + 1)}$ for action $a_{k_i} \in \bar{\mathcal{A}}_i(\ell)$, in which $\bar{N}_{a_i}$ is the number of elements of $\bar{\mathcal{A}}_i(\ell)$ and it is assumed that actions in $\bar{\mathcal{A}}_i(\ell)$ have been reordered sequentially as $k = 1, 2, \dots, \bar{N}_{a_i}$ from the smallest penalty $P_k[i][j]$ to the largest one.

3.6 Overall Algorithm for Our Proposal

The overall algorithm for our proposal is as follows:

1. We set $Q_k[i][j] = 0$ and $P_k[i][j] = 0$ for all $i = 1, 2, \dots, N_m$, $j = 1, 2, \dots, N_a$ and $k = 1, 2, \dots, N_B$ where N_m , N_a and N_B are the numbers of dimensions and elements for the action, and discrete states. Set $TUC_k = 0$, i.e., the number of times to get a reward, and initialize $life_k$, i.e., the life span for forgetting a useless basis function, for all k . In simulation, we set $life_k = 100$.

2. Repeat the following steps until the end condition for learning is met, e.g., a certain number of episodes.
 - (a) Initialize the position of the agent. We make s_0 , an arbitrary discrete state, including the initial state.
 - (b) Initialize a queue and set $E_k[i][j] = 0$ for all i, j, k where $i = 1, 2, \dots, N_m$, $j = 1, 2, \dots, N_a$ and $k = 0, 1, 2, \dots, N_B$.
 - (c) Repeat the following steps until the end condition of an episode is met.
 - i. The agent selects action act_q in discrete state s_k at $epoc = t$ based on the method in Section 3.5.
 - ii. For each j at dimension i , $E_k[i][j] \leftarrow E_k[i][j] + 1$.
 - iii. The rule is added to the queue as stated in Section 3.2.
 - iv. After the agent selects action act'_q , it transits to next state s'_k generated by the method in [12, 18].
 - v. If the agent gets reward rwd , the following updates are used for all states and actions in the queue:
 $Q_k[i][j] \leftarrow Q_k[i][j] + rwd \cdot E_k[i][j]$ and $TUC_k \leftarrow TUC_k + 1$
 If we use Strategy 1, the state that is $TUC_k \geq K$ changes to the fixed mode state.
 - vi. If the agent gets penalty p_ℓ ($\ell = 1, 2, \dots, N_p$) whose value is $pnltty$ for dimension i , for three states, that have times $epoc = t, t - 1, t - 2$, from the state receiving a penalty, the following is done:

$$P_k[l][i][j] \leftarrow P_k[l][i][j] + pnltty \cdot E_k[i][j] \quad (1)$$

If there is a fixed mode state in three states, the state changes to a non fixed mode state.

We execute PRD [12, 18] for the state getting a penalty directly to find a new penalty state.

If a new penalty state is found, the penalty matrix of the rule transiting to the new penalty state is updated by equation (1).

If $\ell = N_p$, go to step 2(c)i. Otherwise, go to step 2a.

- vii. We execute discount by $E_k[i][j] \leftarrow \gamma E_k[i][j]$, and the following, execute $life_k \leftarrow life_k - 1$, $k = 0, 1, 2, \dots, N_B$. If $life_k < 0$, state s_k is eliminated from memory (storage).
- viii. We set $s_k \leftarrow s'_k$ and $epoc = t + 1$. If the end condition of learning is satisfied, the next trial is started. If we use Strategy 2, the state that is $TUC_k > 0$ excluding a penalty state changes to a fixed mode state.
- ix. If we satisfy the end condition about an episode, go to step 2a. Otherwise, go to step 2(c)i.

4 Learning of Biped Walking Robot Waist Trajectory

We discuss a successful application of XoL to the inverted pendulum problem[13], applying it to the waist trajectory generator of a biped robot with a learning controller as shown in Fig.1. The trajectory generator learns a waist trajectory

Fig. 1. Trajectory generation of biped robot

for stable dynamic walking by modifying the waist trajectory for static walking. The waist trajectory for static walking is relatively easily designed. The design for dynamic stable walking is usually formulated as an optimization problem with two-point constraints for a nonlinear dynamic system, and the solution is very difficult.

This learning task has features the following: (1) Sensory input consists of both piecewise continuous variables and discrete variables. Joint angular velocities, for example, are continuous but jump when the robot's swing leg touches the ground. One example of a discrete variable is the index for the single-leg support phase and the two-leg support phase. (2) Generator output consists of continuous waist coordinates. (3) There are multiple types of failure in learning runs. The zero moment point (ZMP), for example, went out from a support polyhedron, inverse kinematics could not be solved, or joint torque exceeded the limit. (4) The task is long-term, i.e., from starting a walk to stopping it after several steps.

This task is therefore a good test bed for determining the effect of the fixed mode state in Section 3.

The simulated robot consists of two legs and a torso. Each leg has 6 degree of freedom (dof) and total dof is 12. Parameters resemble an upper grade elementary school student in size and mass – 1.1 [m] tall and weighing 28.87 [kg]. The control sampling time is 2[ms] and the learning sampling time 20[ms].

4.1 States for Learning

We prepare the following sensory input as states for learning. (1) 10 for the right-leg support gate, and -10 for the left-leg support gate. (2) 10 for the two-leg support phase, and -10 for the single-leg support phase. (3) Coordinate (x, y) of the waist for the support leg. The height of the waist is kept constant. (4) Coordinate (x, y) of ZMP error. (5) Acceleration $(\ddot{x}, \ddot{y})$ of the waist.

Dimensions of the state is $N_s = 8$. Variables from (3) to (5) are scaled with appropriate factors.

4.2 Definition of Actions and Modifying Waist Trajectory

The action specifies the modification values (dx, dy) from the static walking waist trajectory, so the dimension of the action is $N_m = 2$ and selected from $-0.024[\text{m}]$ to $0.024[\text{m}]$, which is discretized to 15 alternatives, i.e., $N_a = 15$. The number of actions is $15^2 = 225$. In our new framework, however, the number of subtasks is reduced to 30.

Learning takes place every 20 ms. We use 3-order B-spline functions to generate a smooth waist trajectory from modified values.

4.3 Rewards and Penalties

Table 1. Definition of penalties

p_ℓ	pnlty	Conditions
0	10	When $\ \text{ZMP error}\ \geq 6.5\text{cm}$
1	10	When $ \text{the joint velocity} \geq 20\pi$
2	10	When the state transits to a penalty state

Table 2. Definition of rewards

Name	rwd	Condition
RWDforGD	10	When the agent reaches to the target time t_f but $Terr \geq \widehat{Terr}$
RWDforSuc	10000	When the agent reaches to the goal time and $Terr < \widehat{Terr}$

We use three types of penalties in Table 1, i.e., $N_p = 3$.

If the agent gets penalty p_ℓ , each of the subrules taken at the last epoc is given penalty $pnlty$ and is decided using λ for whether it should be classified as a penalty rule. This helps accelerate learning. In general, however, we must be careful when using this method because it has the possibility of generating too many penalties. Threshold λ controls this problem.

To define rewards, we use the following cost function:

$$\begin{aligned}
Terr = & \frac{1}{t_f} \int_0^{t_f} \left\{ \left(\frac{\|\mathbf{Zerr}(t)\|}{zmpLimit} \right)^2 t \right\} dt + 50 (|Zerr_x(t_f)| + |Zerr_y(t_f)|) \\
& + \frac{1}{t_f} \sqrt{1000 \int_0^{t_f} \|\hat{\mathbf{q}}(t) - \mathbf{q}(t)\|^2 dt + \frac{1}{1000} \int_0^{t_f} \|\mathbf{u}(t)\|^2 dt} + 10 \frac{\sum_{i=1}^{dof} |\hat{\mathbf{q}}_i(t_f) - \mathbf{q}_i(t_f)|}{dof} \\
& + 10 \frac{\sum_{i=1}^{dof} |\hat{\dot{\mathbf{q}}}_i(t_f) - \dot{\mathbf{q}}_i(t_f)|}{dof} \tag{2}
\end{aligned}$$

where t_f is target walking time, $Zerr_x$ and $Zerr_y$ are x and y components of ZMP error, and $\mathbf{u}(t) \in \mathcal{R}^{dof}$ is a joint torque vector. $zmpLimit$ is the allowable ZMP error limit, i.e., $zmpLimit = 0.065$ in simulation, and $dof = 12$.

When the agent reaches to target time t_f , we calculate $\widehat{Terr}$ that is the estimation of final cost $Terr$ as follows:

$$Initialization: \widehat{Terr} = 1.05Terr \quad (3)$$

$$After\ initialization: \widehat{Terr} = \begin{cases} 0.5\widehat{Terr} + 0.5Terr & (Terr < \widehat{Terr}) \\ 0.9\widehat{Terr} + 0.1Terr & (Terr \geq \widehat{Terr}) \end{cases} \quad (4)$$

Rewards are defined in Table 2. RWDforSuc is given to the agent when it reaches the target walking time the first time or, after that, when $Terr < \widehat{Terr}$. In this case, the counter of success is updated as $TUC + 1$. If the agent reaches it but $Terr \geq \widehat{Terr}$, then the agent gets RWforGD.

5 Simulation Results

5.1 Learning Schedule

Since our learning task is over rather long-term, we schedule it as follows:

(1) in the initial stage, target walking time t_f is set to $t_f = Tstep$, where $Tstep$ is one step time. (2) t_f is extended by $Tstep/2$ after the agent receives RWDforSuc LT times until t_f reaches full term t_w .

The robot starts from a two-leg standing pose, walks 5 steps, and stops with a two-leg standing pose, again, as shown in Fig.1. It takes $t_w = 11.25[s]$, including preparation time.

The agent could not walk using the initial walking trajectory designed for stable static walking. The following results are averages for 15 trials with different random seeds.

5.2 Simulation (1) : Effect of Strategy 1 for Fixed Mode State

We first conducted simulation to confirm the effect of Strategy 1 for the fixed mode state, where we use $LT = 15$ and set $K = 1, 3, 5, 10, 15, 20$ and ∞ . Fig.2 shows normalized total learning time and the number of episodes for each K . We confirm an improved number of episodes and learning time compared to $K = \infty$, meaning that we do not use the fixed mode state. In $K = 15(= LT)$, we see a two times higher performance. For $K = 1$, the number of episodes and learning time are near the case of $K = \infty$.

Fig.3 shows the ratio of the number of fixed mode states to N_B for each K . We confirmed a large change in the ratio of the value of K and the change in learning speed. For $K = 1$, states change to fixed mode very easily, so the agent has difficulty in finding undiscovered states, degrading performance.

5.3 Simulation (2) : Effect of Strategy 2 for Fixed Mode State

We conducted simulations to confirm the effect of Strategy 2 for fixed mode states for $LT = 1, 3, 5, 10$ and 15 . Fig.4 shows the number of episodes for each

Fig. 2. Strategy1: Total learning time and number of episodes **Fig. 3.** Strategy1: Ratio of number of fixed mode states to N_B

Fig. 4. Strategy2: Number of episodes for LT

Fig. 5. Strategy2: Total learning time for LT

LT , and Fig.5 is the total learning time. Red (ST2) corresponds to Strategy 2 and blue (ST0) to a case in which no fixed mode state is used. We confirmed the effectiveness of Strategy 2 for $LT = 1$ and 3 in both the number of episodes and total learning time.

Fig.6 shows the ratio of the number of fixed mode state to N_B and that the ratio is almost constant for Strategy 2 over a large range of LT , so Strategy 2 is easy to use in practice compared to Strategy 1.

Fig. 6. Strategy2: Ratio of number of fixed mode states to N_B

6 Conclusions

We have proposed new exploitation-oriented learning (XoL) having *fixed mode state* shifted from normal when the state has already received sufficient reward. We used online PS to get rewards efficiently in long-term tasks.

We have adapted our method to generating a biped robot waist trajectory. We have discussed simulation results and confirmed our method's effectiveness. Now we are studying the relationship between K and LT to determine how to choose appropriate K .

We plan to apply our method to other control problems, e.g., the bicycle balancing problem[14] nonholonomic systems[6, 9]. We also plan to extend our method to use with a real-world biped robot and explore real-world applications.

This work was supported by Grant-in-Aid for Scientific Research(C) 21560275.

References

1. Chrisman, L.: Reinforcement Learning with perceptual aliasing: The Perceptual Distinctions Approach. Proc. of the 10th National Conf. on Artificial Intelligence, 183–188 (1992)
2. Hasemi, K., Suyari, H.: A Proposal of algorithm that reduces computational complexity for Online Profit Sharing. Report of the Institute of Electronics, Information and Communication Engineers. NC-105(657), 103–108 (2006) (in Japanese)

3. Kimura, H., Kobayashi, S.: An analysis of actor/critic algorithm using eligibility traces: reinforcement learning with imperfect value function. *Proc. of the 15th Int. Conf. on Machine Learning*, 278–286 (1998)
4. Goldberg, D. E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional (1989)
5. Heidrich-Meisner, V. and Igel, C.: *Evolution Strategies for Direct Policy Search. Parallel Problem Solving from Nature (PPSN X)*. 5199 of LNCS, 428–437, Springer-Verlag (2008)
6. Ikeda, K.: Exemplar-Based Direct Policy Search with Evolutionary Optimization. *Proc. of 2005 Congress on Evolutionary Computation CEC2005*, 2357–2364 (2005)
7. Matsui, T., Goto, T., Izumi, K.: Acquiring a Government Bond Trading Strategy Using Reinforcement Learning. *J. of Advanced Computational Intelligence and Intelligent Informatics*. 13(6), 691–696 (2009)
8. Merrick, K., Maher, M. L.: Motivated Reinforcement Learning for Adaptive Characters in Open-Ended Simulation Games. *Proc. of the Int. Conf. on Advanced in Computer Entertainment Technology*, 127–134 (2007)
9. Miyamae, A., Sakuma, J., Ono, I., Kobayashi, S.: Instance-based Policy Learning by Real-coded Genetic Algorithms and Its Application to Control of Nonholonomic Systems. *J. of The Japanese Society for Artificial Intelligence*. 24(1), 104–115 (2009) (In Japanese)
10. Miyazaki, K., Yamamura, M., Kobayashi, S.: On the Rationality of Profit Sharing in Reinforcement Learning. *Proc. of the 3rd Int. Conf. on Fuzzy Logic, Neural Nets and Soft Computing*, 285–288 (1994)
11. Miyazaki, K., Kobayashi, S.: Reinforcement Learning for Penalty Avoiding Policy Making. *Proc. of the 2000 IEEE Int. Conf. on Systems, Man and Cybernetics*, 206–211 (2000)
12. Miyazaki, K., Kobayashi, S.: A Reinforcement Learning System for Penalty Avoiding in Continuous State Spaces. *J. of Advanced Computational Intelligence and Intelligent Informatics*. 11(6), 668–676 (2007)
13. Miyazaki, K., Kobayashi, S.: Exploitation-Oriented Learning PS-r[#]. *J. of Advanced Computational Intelligence and Intelligent Informatics*. 13(6), 624–630 (2009)
14. Randløv, J., Alstrøm, P.: Learning to Drive a Bicycle Using Reinforcement Learning and Shaping. *Proc. of the 15th Int. Conf. on Machine Learning*. 463–471 (1998)
15. Stone, P., Sutton, R. S., and Kuhlmann, G.: Reinforcement Learning toward RoboCup Soccer Keepaway. *Adaptive Behavior*. 13(3), 165–188 (2005)
16. Sutton, R. S., Barto, A. G.: *Reinforcement Learning: An Introduction*. A Bradford Book. MIT Press (1998)
17. Sutton, R. S., McAllester, D., Singh, S. P., Mansour, Y.: Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems*. 12, 1057–1063 (2000)
18. Watanabe, T., Miyazaki, K., Kobayashi, H.: A New Improved Penalty Avoiding Rational Policy Making Algorithm for Keepaway with Continuous State Spaces. *J. of Advanced Computational Intelligence and Intelligent Informatics*. 13(6), 675–682 (2009)
19. Yoshimoto, J., Nishimura, M., Tokita, Y., Ishii, S.: Acrobot control by learning the switching of multiple controllers. *J. of Artificial Life and Robotics*. 9(2), 67–71 (2005)