Artif Life Robotics (2001) 5:202–206

ORIGINAL ARTICLE

Hiroki Tamakoshi · Shin Ishii

Multiagent reinforcement learning applied to a chase problem in a continuous world

Received: October 30, 2000 / Accepted: May 30, 2002

Abstract Reinforcement learning (RL) is one of the methods of solving problems defined in multiagent systems. In the real world, the state is continuous, and agents take continuous actions. Since conventional RL schemes are often defined to deal with discrete worlds, there are difficulties such as the representation of an RL evaluation function. In this article, we intend to extend an RL algorithm so that it is applicable to continuous world problems. This extension is done by a combination of an RL algorithm and a function approximator. We employ Q-learning as the RL algorithm, and a neural network model called the normalized Gaussian network as the function approximator. The extended RL method is applied to a chase problem in a continuous world. The experimental result shows that our RL scheme was successful.

Key words Multiagent · Reinforcement learning · Continuous system · Function approximation · Normalized Gaussian network

1 Introduction

Human beings have long been considering models of intelligent beings. Recently, we have come to recognize the importance of communication among such beings in order to organize a complex society. We call these beings *agents*; we are interested in a model of a society consisting of several or many agents, i.e., a multiagent system.

In the research field of multiagent systems, there are two main approaches to constructing intelligent agents. One is to design the behaviors of the agents by hand, and the other is to let the agents learn appropriate behaviors. In this article, we take the latter approach. Although agents can act and communicate with each other in a given environment, they do not know the meaning of actions or communication prior to learning. They learn the meaning through their experiences in interacting with the environment.

In this study, we consider a multiagent cooperation problem in which two agents work together. To let agents learn how to cooperate, we use a *reinforcement learning* (RL) scheme. In RL schemes, an agent learns how good or bad an action is through experience. However, since RL schemes were originally developed in single-agent environments, the convergence of learning in multiagent environments has not yet been theoretically guaranteed. On the other hand, a lot of case studies have shown encouraging results for the applicability of RL to multiagent environments.

In a realistic world, the state space is continuous, and an agent takes a continuous action in that space. However, conventional RL schemes assume that both state and action spaces are discrete, and therefore applications of such RL schemes to continuous-world problems involve difficulties. One is the representation of the RL evaluation function. In order to represent this function efficiently, we employ a *normalized Gaussian network* (NGnet) as a function approximator in this study. The NGnet is a kind of layered neural network. This model loosely partitions the input space using normalized Gaussian functions, and each local unit approximates the output within its partition. Since the task considered in this study has a spatially local structure, the NGnet is suitable for approximating the evaluation function.

In order to stabilize learning, we conduct an annealing procedure for the learning coefficient. The merit of the annealing procedure is experimentally confirmed in this study.

H. Tamakoshi · S. Ishii (✉)
Nara Institute of Science and Technology, Takayama 8916-5, Ikoma, Nara, Japan
Tel. +81-743-72-5984; Fax +81-743-72-5989
e-mail: ishii@is.aist-nara.ac.jp

S. Ishii
CREST, Japan Science and Technology Corporation, Japan

This work was presented in part at the Fifth International Symposium on Artificial Life and Robotics, Oita, Japan, January 26–28, 2000

2 Reinforcement learning scheme

2.1 General preliminaries

Before introducing our reinforcement learning scheme, we describe the notation. Let us first consider a single agent, where s is an environment state that the agent perceives, a is an action that the agent takes, S is the set of s, A is the set of a, $\pi: S \mapsto A$ is a policy function that determines an action for each environment state, r is a reward ($\in \mathbb{R}$), and $R: S \mapsto \mathbb{R}$ is a reward function that determines a reward for each state. It is assumed that the agent is able to perceive environment s around him. Then the agent takes an action a which is determined by the policy function π. It is also assumed that a reward r is given to the agent. The reward represents how good or bad the state is; a larger reward implies a better state.

The agent interacts with the environment as follows. (i) The agent takes an action a determined by the policy function π for the current state s. (ii) Then the agent interacts with the environment, perceives the next state s', and gets a reward r generated by the reward function R for state s'. Processes (i) and (ii) are repeated until the agent achieves its goal. The purpose of the agent is to maximize the sum of the rewards. The reward function is defined such that the sum of the rewards is large when the agent achieves a goal in an efficient manner.

2.2 Q-learning

Our RL scheme is based on Q-learning.[1,2] In Q-learning, an action-value function, or a Q-function, $Q(s, a)$, is defined as the estimated sum of rewards if action a is taken at state s, and the subsequent actions are determined by the best policy. If the optimal action-value function $Q^*(s, a)$ is known, the agent can determine the optimal action by $\arg\max_a Q^*(s, a)$ for any state s.

Q-learning is simply stated as follows: after a state transition occurs from s to s' by taking an action a, and a reward is given by $r = R(s')$, then the following renewal of $Q(s, a)$ takes place.

$$Q(s,a) \leftarrow (1-\eta)Q(s,a) + \eta_t\left\{r + \beta \max_{a'} Q(s',a')\right\} \tag{1}$$

where η_t and β are a learning coefficient and a discount factor, respectively.

The update rule in Eq. 1 influences only a single state–action pair (s, a). Since the number of possible state–action pairs is infinite in a continuous state–action space, the Q-learning algorithm employing a Q-function defined for a discrete state–action space becomes computationally intractable. In order to overcome this difficulty, a function approximator, e.g., a neural network, is used to represent a Q-function defined for the continuous state–action space. Here, we use the NGnet to approximate the Q-function.

2.3 NGnet and the learning algorithm

The NGnet[3] is a network of normalized Gaussian functions. The model loosely partitions the input space, and the output is given as the sum of the values given for the partitions. The model is described as

$$\boldsymbol{y} \equiv \sum_{i=1}^{M} \mathcal{N}_i \boldsymbol{w}_i \equiv \sum_{i=1}^{M} \left(\frac{g_i(\boldsymbol{x})}{\sum_{k=1}^{M} g_k(\boldsymbol{x})} \right) \boldsymbol{w}_i \tag{2}$$

$$g_i(\boldsymbol{x}) \equiv \frac{1}{(2\pi)^{\frac{N}{2}} \sigma_i^N} \exp\left(-\frac{|\boldsymbol{x} - \boldsymbol{\mu}_i|}{2\sigma_i^2} \right) \tag{3}$$

where $\boldsymbol{x}$ is an N-dimensional input vector, and $\boldsymbol{y}$ is a D-dimensional output vector. There are M normalized Gaussian units. g_i is the i-th Gaussian function which has an N-dimensional center vector $\boldsymbol{\mu}_i$ and a variance σ_i^2, and $\boldsymbol{w}_i$ is a D-dimensional vector. The model parameters are $\boldsymbol{\theta} = \{\boldsymbol{\mu}_i, \sigma_i^2, \boldsymbol{w}_i | i = 1, \ldots, M\}$.

We use a *gradient descent* method to let the NGnet learn the relation between input $\boldsymbol{x}$ and target output $\boldsymbol{d}$. The learning is done by

$$E \equiv \frac{1}{2}|\boldsymbol{d} - \boldsymbol{y}|^2 = \frac{1}{2}\sum_{j=1}^{D}(d_j - y_i)^2 \tag{4}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \rho \frac{\partial E}{\partial \boldsymbol{\theta}} \tag{5}$$

where E is the squared error between the NGnet output $\boldsymbol{y}$ for input $\boldsymbol{x}$ and the target output $\boldsymbol{d}$, and ρ is a learning coefficient. The partial differentiation for each parameter is calculated as

$$\frac{\partial E}{\partial \boldsymbol{\mu}_i} = \frac{(\boldsymbol{x} - \boldsymbol{\mu}_i)\mathcal{N}_i}{\sigma_i^2} \sum_{j=1}^{D} (y_j - d_j)(w_{ij} - y_j) \tag{6}$$

$$\frac{\partial E}{\partial \alpha_i} = \frac{\mathcal{N}_i}{2}\left(|\boldsymbol{x} - \mu_i|^2 - \frac{N}{\alpha_i} \right) \sum_{j=1}^{D} (d_j - y_j)(w_{ij} - y_j) \tag{7}$$

$$\frac{\partial E}{\partial w_{ij}} = (y_j - d_j)\mathcal{N}_i \tag{8}$$

where $\alpha_i \equiv \sigma_i^{-2}$.

2.4 RL algorithm with NGnet

The Q-function $Q(s, a)$ defined on the continuous state–action space is approximated by the NGnet. After each transition of the agent, the NGnet is trained using the target output in Eq. 1 for input (s, a).

We also use an annealing procedure for the learning coefficient η_t of Q-learning. The annealing procedure somehow forms a stochastic approximation. In the experiment,

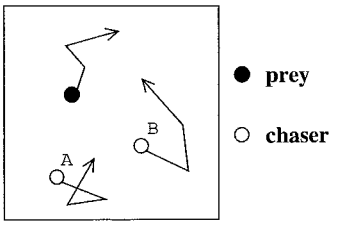

Fig. 1. A multiagent chase problem

the learning coefficient reduces exponentially, $\eta_t \equiv \gamma\eta_{t-1}$, where t is the number of learning episodes, and $0 < \gamma < 1$ is a decay constant.

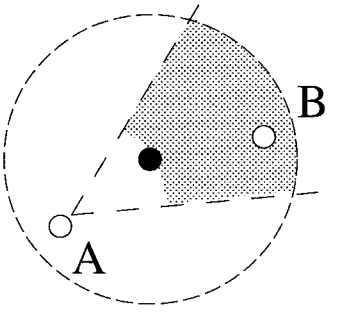

Fig. 2. Capture zone. When chaser B is in the *gray area*, there is the possibility of capturing the prey

3 Result

3.1 Task definition

We consider a chase problem in a continuous world (Fig. 1), which is a natural extension of the problem defined in a discrete space.[4] The world is a 2D plane and has conjunctive edges.

The system consists of three agents: two chasers, A and B, move cooperatively to attack a prey from both sides. Each agent is assumed to have no volume in space, i.e., it is just a point. The prey does not learn anything, but has an a priori simple strategy. At each discrete time step, it randomly moves to another position that is limited within a unit circle around the current position, and avoids approaching the chasers. Each chaser acts as follows. (1) It perceives the relative positions of the prey and the other chaser with respect to itself, which is represented as $s \equiv \{s^p, s^c\}$, where s^p and s^c denote two-dimensional location vectors for the prey and the other chaser, respectively. (2) It takes an action $a = \arg\max_a Q(s, a)$, where a is a two-dimensional vector and is restricted to be $|a| \leq 1$. In other words, at each time step, each agent is allowed to move within a unit circle around the current position. When the agent takes an action, a small Gaussian noise is added to the radial of the action selected. This noise represents the uncertainty of the actual action. (3) According to the actual action, the state changes. After each agent receives a reward corresponding to the new state, its own Q-function is updated based on Q-learning.

There is a movement order in this system. First the prey moves, then chaser A moves, and finally chaser B moves. These processes are repeated until a goal is achieved.

3.2 Goal state and reward

The goal and the reward function are defined as follows. We define a "capture zone" that is a unit circle whose center is the current prey position.

When neither of the chasers are in the capture zone, there is no possibility of capturing the prey, and the chasers receive a reward of -0.2. If both of the chasers are in the capture zone, there are two cases, as described below.

Let D_{xy} denote the distance between two agents x and y, where x (or y) is p, A, or B for the prey, chaser A, or chaser B, respectively. If $D_{pA} < D_{AB}$ and the angle between the vector from A to the prey and the vector from A to B is less than $\pi/6$ (case 1), there is the possibility of capturing the prey. This subzone in the capture zone is shown as a gray area in Fig. 2. If this condition is not satisfied (case 2), the reward is -0.1. For case 1, there are two subcases. At possibility $\exp\{-(D_{pA} + D_{pB})\}$, the goal is achieved, and a reward of 1.0 is given to the both chasers (case 1.1). If the goal is not achieved, the reward is 0.0 (case 1.2). If case 1.1 occurs, a single episode ends; otherwise it continues.

3.3 Solving $\arg\max_a Q(s, a)$

In our learning scheme, it is difficult to obtain $\arg\max_a Q(s, a)$ because there is an infinite number of action candidates for the current state s. Several methods have been proposed[5] to obtain a semioptimal action from an infinite number of candidates. We use the following method in this study.

For an action candidate $a_0(= (0, 0)) \in A$, we prepare a circle in the action space. The center and radius of the circle are a_0 and $I_0(= 0.95)$, respectively. The circle is partitioned into mesh grids, and the Q-function is evaluated on the grid points. The grid point whose Q-function value is the largest is the next candidate, a_1. In the next phase, we prepare a smaller circle with center is a_1 and radius is $I_1 = 0.05I_0$, and the grid point whose Q-function value is the largest obtained in the new circle. This procedure is repeated until radius I_n becomes small enough. The obtained a_n is the action that the agent takes. Note that this method only obtains in a locally optimal action.

3.4 Experimental results

The experiment was conducted under the following conditions.

1. World size: 7×7.
2. Q-learning parameters: $\eta = 0.1$, $\beta = 0.9$, and annealing coefficient: $\gamma = 0.996$.
3. NGnet parameters: $M = 50$, and $\rho = 0.1$

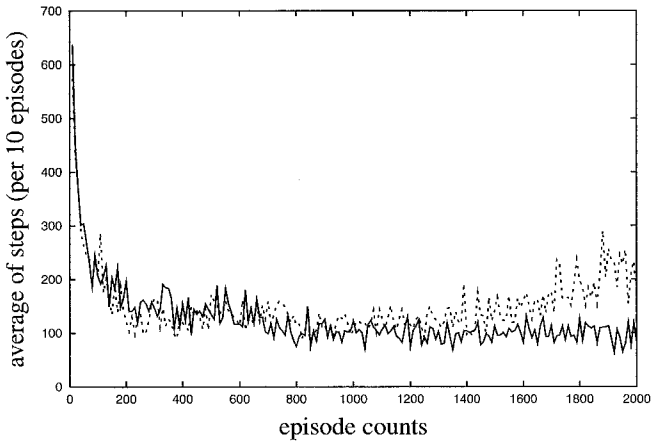

Fig. 3. Learning process with annealing (*solid lines*) and without annealing (*dashed lines*). The learning curve corresponds to the steps toward a goal averaged over ten learning episodes. Each line denotes the learning curve averaged over five learning trials, i.e., for different initial conditions

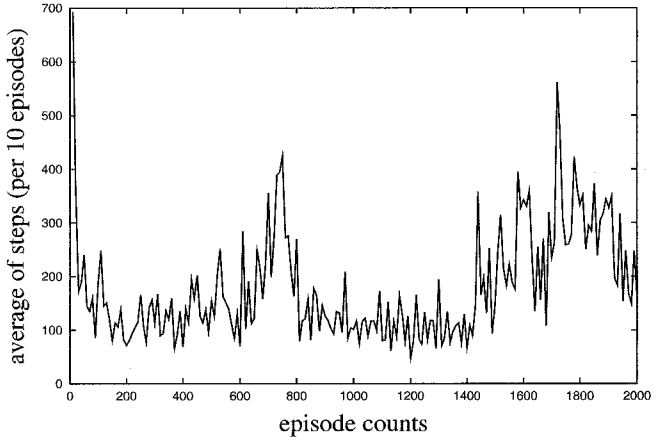

Fig. 4. A learning process without annealing

Figure 3 shows the steps to the goal plotted against the number of training episodes. In each episode, the initial states of the agents are randomly prepared in the world. The solid and dashed lines denote the results with and without the annealing process, respectively. The learning curve for each condition is obtained as the steps averaged over ten learning episodes. We conducted five learning trials by varying the initial learning conditions, and Fig. 3 shows their average. This result shows that both agents have successfully learned appropriate behaviors.

Figure 4 shows a single learning trial without the annealing. As this figure shows, the learning often becomes unstable without the annealing process. Accordingly, it is found that the annealing procedure is important for stabilizing the learning.

Figure 5 shows examples of actual movements of the agents after learning. The thin, medium, and thick lines denote the movements of the prey, chaser A, and chaser B, respectively. The center of the two arcs denotes the position where the prey is captured. We can see that the two chasers

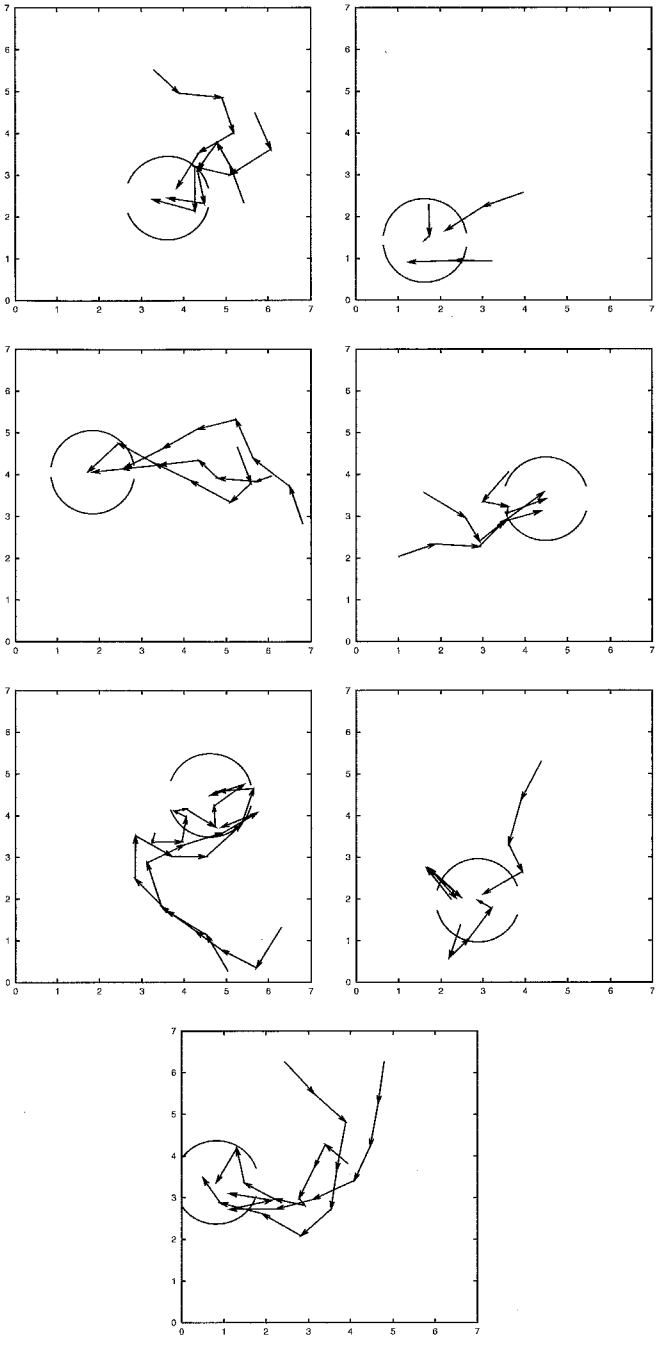

Fig. 5. Trajectory examples. The *thin*, *medium*, and *thick lines* denote the movements of the prey, chaser A, and chaser B, respectively. The center of the two arcs in each case denotes the position where the prey is captured

first approach the prey and then exhibit cooperative behavior to capture it.

Finally, we show the Q-function of chaser B after learning. Since the Q-function is defined for six-dimensional continuous space, it is difficult to visualize the function in the whole space. Thus, we restrict the situations to typical ones, and evaluate the Q-function for those situations. Consider the situations described by Fig. 6. The biggest circle is an unit circle whose center is the prey's position. Chaser A

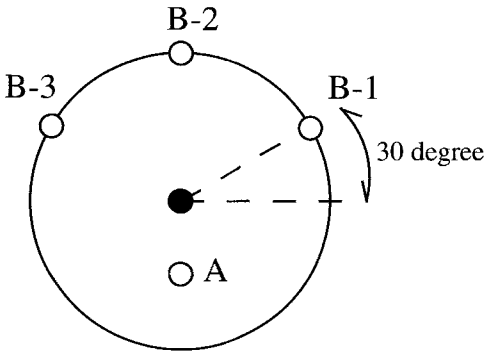

Fig. 6. Sample points for Q-function snapshots. The circle denotes the unit circle around the prey (*black point*). B-1, B-2, and B-3 denote the three possible positions for chaser B

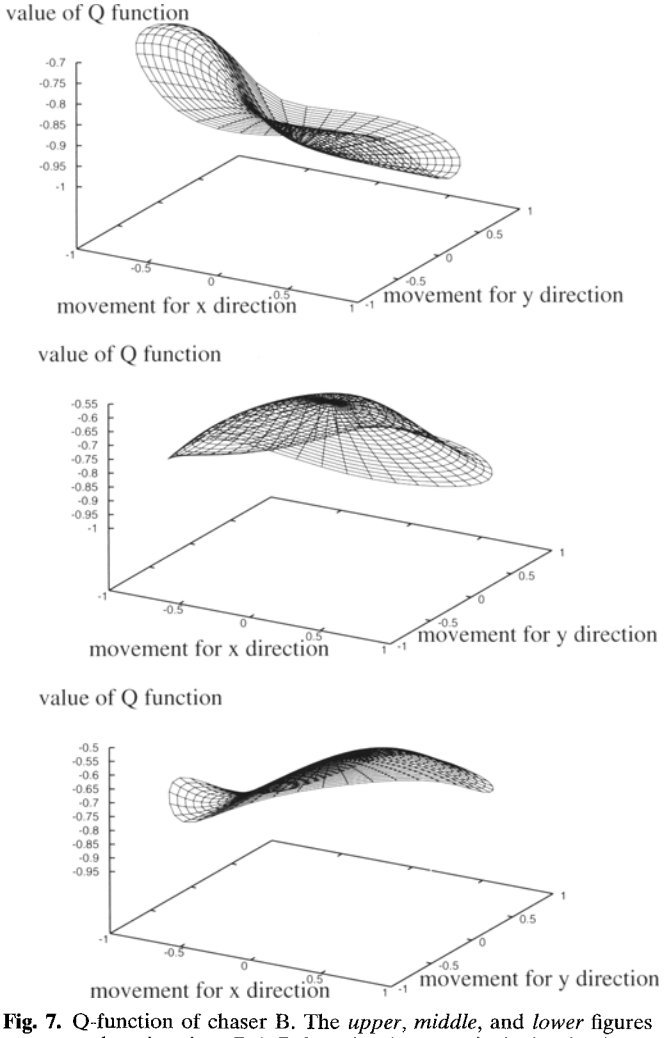

Fig. 7. Q-function of chaser B. The *upper*, *middle*, and *lower* figures correspond to situations B-1, B-2, and B-3, respectively, in Fig. 6

locates to 0.5 south from the prey. We consider three situations for chaser B, as shown in Fig. 6.

Figure 7 shows the Q-function of chaser B for each of the three situations. The x and y axes denote the action space. The first, second, and third graphs describe the Q-function at states B-1, B-2, and B-3, respectively. In the figure for situation B-1, the Q-function value is largest when the movement direction is west. By such a movement, chaser B will position itself north of the prey, which is an appropriate position to capture the prey cooperatively. Similarly, chaser B will move to the south for situation B-2, and east for situation B-3. These movements are the optimal ones to capture the prey cooperatively.

4 Concluding remarks

Our study is basically an extension of an existing study dealing with a discrete state–action space.[4]

There have been several studies that extend state or action space to a continuous one.[5–7] However, their methodology for representing the continuous state space is based on a fine discretization of the space, or on an interpolation over the discrete representation of the space. On the other hand, our study intends to deal essentially with the space continuity using a function approximator.

We introduced a Q-learning scheme that employed an NGnet as a function approximator, which led to a successful result in the experiment. We consider that our method is appropriate for dealing with multiagent problems that have continuous state and action spaces.

Tan[4] examined the effect of communication between agents, which is an important aspect of multiagent systems. For human beings, communication is used not only to transfer information, but also to model and predict the behavior of partners. We will consider such an aspect in future studies.[8]

References

1. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press, Cambridge
2. Watkins CJCH, Dayan P (1992) Mach Learn J 8(3/4):279–292
3. Moody J, Darken CJ (1989) Neural Comput 1:281–294
4. Tan M (1993) Proceedings of the 10th International Conference on Machine Learning, p 330–337
5. Baird LC, Klopf AH (1993) Technical Report WL-TR-93-1147, Wright–Patterson Air Force Base, Ohio, Wright Laboratory
6. Harmon ME, Baird LC (1996) Technical Report WL-TR-1065, Wright–Patterson Air Force Base, Ohio, Wright Laboratory
7. Munos R (1998) Proceedings of the 10th European Conference on Machine Learning, Lecture Notes in Artificial Intelligence, vol 1398, p 394–405
8. Tamakoshi H, Ishii S (2002) 4th Asia-Pacific Conference on Simulated Evolution and Learning