Q-Value Based Particle Swarm Optimization for Reinforcement Neuro-Fuzzy System Design

Yi-Chang Cheng
Dept. of Electrical Engineering
National Chiao Tung University
HsinChu, Taiwan
ottocheng.ece94g@nctu.edu.tw

Sheng-Fuu Lin
Dept. of Electrical Engineering
National Chiao Tung University
HsinChu, Taiwan
sflin@mail.nctu.edu.tw

Chi-Yao Hsu
Dept. of Electrical Engineering
National Chiao Tung University
HsinChu, Taiwan
hsuchyao@yahoo.com.tw

Abstract—This paper proposes a combination of particle swarm optimization (PSO) and Q-value based safe reinforcement learning scheme for neuro-fuzzy systems (NFS). The proposed Q-value based particle swarm optimization (QPSO) fulfills PSO-based NFS with reinforcement learning; that is, it provides PSO-based NFS an alternative to learn optimal control policies under environments where only weak reinforcement signals are available. The reinforcement learning scheme is designed by Lyapunov principles and enjoys a number of practical benefits, including the ability of maintaining a system's state in a desired operating range and efficient learning. In the QPSO, parameters on a NFS are encoded in a particle evaluated by Q-value. The Q-value cumulates the reward received during a learning trial and is used as the fitness function for PSO evolution. During the trail, one particle is selected from the swarm; meanwhile, a corresponding NFS is built and applied to the environment with an immediate feedback reward. The applicability of QPSO is shown through simulations in single-link and double-link inverted pendulum system.

Keywords- ***Neuro-fuzzy system, particle swarm optimization, reinforcement learning, Q-learning.***

I. INTRODUCTION

In recent years, the application of neuro-fuzzy systems in control engineering has become a popular research topic [1]-[3]. In general, the way of tuning the parameters on a NFS can be divided into two categories: supervised learning [4] and reinforcement learning [5].

First, supervised learning is a machine learning technique for updating its parameters from training data. The training data is composed of pairs of inputs, and desired outputs. The object of the supervised learning is to predict the output value of the NFS for any valid input data after its parameters have been trained by a number of training data. However, for many control tasks, training data are usually difficult or too costly, or even not accessible. As a result, reinforcement learning is more practicable than supervised learning in many occasions.

In reinforcement learning, the agent receives from its environment a reinforcement signal at each time step. This signal could be either a reward or a punishment. Meanwhile, the agent explores actions from the action set, and finds out which action yields the greatest reward. To solve reinforcement problem, temporal difference (TD) [6]-[8] is one of the most common method. In TD learning, learners don't have to wait until the end of a trial; instead, TD methods need wait only one time step. This is crucial for applications that have very long trials or tasks that are continuous and have no trials at all. Q-learning [9] is a powerful and easy-implementing TD-based approach. It is a reinforcement learning technique that works by updating a simple action-value iteration function. This function gives the measurement of taking a given action in a given state.

Besides TD methods, many evolutionary algorithms such as PSO [10]-[12], genetic algorithm (GA) [13], evolutionary programming [14], and evolution strategies [15], are popular for solving reinforcement learning tasks. These learning procedures are based on populations made of individuals with specific behaviors similar to certain biological phenomena. Individuals keep exploring the solution space and exploiting information between individuals while evolution proceeding. In general, by means of exploring and exploiting, evolutionary algorithms are less likely to be trapped at the local optimum. Many researches on using evolutionary algorithm for solving reinforcement learning tasks have been proposed recently [16]-[17]. In [16], authors propose a swarm intelligence based reinforcement learning (SWIRL) method to train artificial neural networks (ANN). Authors apply ant colony optimization to select ANN topology and apply the PSO to adjust ANN connection weights. In [17], Lin and Hsu present a reinforcement hybrid learning algorithm (R-HELA) combining the compact GA (CGA) [18] and the modified variable-length GA (VGA) [19] on recurrent wavelet-based NFS. A counter is used to accumulate the time steps until the control task fails and the accumulated values are fed into individuals as fitness functions. Lin and Hsu's model is very effective; however, its fitness function only indicates how long can the controller work well instead of measuring how soon the system can meet the control goal, which is also very important in reinforcement learning. There is also a growing interest in combining the advantages of evolutionary algorithms and TD-based reinforcement learning [20]-[21]. In [20], a TD and GA based reinforcement learning (TDGAR) is proposed. Authors propose a neural structure composed of two feedforward networks for reinforcement learning, the critic network and the action network. The critic network predicts the external signal provides a more informative internal signal to the action network. The action network uses GA to determine the output of the learning system. The weight update rule for the hidden layer of the critic network is based on error backpropagation. In [21], an on-line clustering and Q-value based GA reinforcement learning for fuzzy system (CQGAF) is proposed. In one generation CQGAF learning, one individual is applied to the environment to estimate the fitness function, Q-value, and Q-values of other individuals are updated by eligibility trace. The GA operation is performed by the end of each trial and creates a new generation of individuals. In [22], authors proposed a recurrent wavelet-based NFS with a reinforcement group cooperation-based symbiotic evolution (R-GCSE) algorithm. In [22], a population is divided to several groups. The R-GCSE has a good ability of parameter learning by adopting the concept that each group formed by a set of chromosomes cooperates with other groups to generate better chromosomes.

Although the aforementioned reinforcement learning methods work well in many applications, there is an issue remains to be solved. No fitness function in these methods indicates how soon the learning agents can control the system's state into a set of goal states. Sure there is no need to define the fitness function that way if there is no guidance provided to the controller of how to maintain the system's state in a desired operating range. As a result, in the proposed QPSO, we use the concept of Lyapunov design [23] for constructing safe reinforcement learning agents. We also manipulate our fitness function so that it can indicate how soon the controller achieves its control goal. In our method, a TSK-type NFS [24] is incorporated in the learning system for solving control tasks. The Q-learning is adopted while a particle is applied to the environment for a learning trial. In one generation of the QPSO learning, if there are s particles, s trials are taken to compute the corresponding Q-value of each particle. It allows each particle to fully exploit the information carried on other particles in one generation.

To conclude, the advantages of the QPSO can be shown from that it provides an alternative for Q-learning to solve reinforcement learning problem in one hand, and it extends the applicability of the PSO into reinforcement environment on the other. Moreover, another advantage of the QPSO can be seen from the reliable initial learning performance due to the Lyapunov design of learning agents. These advantages will be verified through simulations.

This paper is organized as follows. Section II presents an overview of the TSK-type NFS, the PSO and the Q-learning. In section III we describe the Lyapunov design for the reinforcement learning agents. This is followed by the algorithm of the QPSO in section IV. In section V, the QPSO is applied to single-link and double-link inverted pendulum system. Finally, conclusions are summarized in section VI.

II. RELATED WORKS

Basic concepts of a TSK-type NFS are first introduced first at section II.A. Followed by the introduction of original PSO and some of its improvement at section II.B. The convergence of PSO is also shown here. The concepts of Q-learning are introduced at section II.C.

A. TSK-Type Neuro-Fuzzy System

A TSK-type NFS is composed of fuzzy IF-THEN rules that can be presented in the following general the form:

$$\begin{aligned}&\text{IF } x_1 \text{ is } A_{1j}(m_{1j},\ \sigma_{1j}) \text{ and } x_2 \text{ is } A_{2j}(m_{2j},\ \sigma_{2j}) \text{ ... and } x_n \text{ is } A_{nj}(m_{nj},\ \sigma_{nj}),\\ &\text{THEN } y' = w_{0j} + w_{1j}x_1 + ... + w_{nj}x_n.\end{aligned} \quad (1)$$

where m_{ij} and σ_{ij} represent the mean and deviation of a Gaussian membership function for i-th dimension and j-th rule node. It's a four-layer structure described as follows:

Layer 1 (Input Node): No function is performed in this layer. The node only transmits input values to layer 2:

$$u_i^{(1)} = x_i\,. \quad (2)$$

Layer 2 (Membership Function Node): Nodes in this layer calculate the membership value specifying the degree to which an input value belongs to a fuzzy set. For an external input, the Gaussian membership is defined by:

$$u_{ij}^{(2)} = \exp\left(-\frac{\left[u_i^{(1)} - m_{ij}\right]^2}{\sigma_{ij}^2}\right), \quad (3)$$

where $u_{ij}^{(2)}$ is the output of second layer with i-th input dimension and j-th rule node; m_{ij} and σ_{ij} are , respectively, the mean and the deviation of the Gaussian membership function with j-th rule node and i-th input dimension.

Layer 3 (Rule Node): The objective of this layer is performing fuzzy AND operation. The product operation is utilized to determine the firing strength of each rule. The function of each rule is:

$$u_j^{(3)} = \prod_i u_{ij}^{(2)}\,. \quad (4)$$

Layer 4 (Consequent Node): The input to a node in layer 4 is the output delivered from layer 3 and other inputs from layer 1. The function of each node is:

$$u_j^{(4)} = u_j^{(3)}(w_{0j} + \sum_{i=1}^{n} w_{ij}x_i)\,. \quad (5)$$

where w_{ij} are the corresponding parameters of the consequent part and M_k is the number of fuzzy rules.

Layer 5 (Output Node): Nodes in this layer correspond to output variables. Each node integrates all the outputs layer 3 and 4 and acts as a defuzzifier with

$$y = u^{(5)} = \frac{\sum_{j=1}^{M_k} u_j^{(4)}}{\sum_{j=1}^{M_k} u_j^{(3)}} = \frac{\sum_{j=1}^{M_k} u_j^{(3)}(w_{0j} + \sum_{i=1}^{M_k} w_{ij}x_i)}{\sum_{j=1}^{M_k} u_j^{(3)}}\,. \quad (6)$$

B. Particle Swarm Optimization

PSO is first introduced by Kennedy and Eberhart in 1995 [10]. It's one of the most powerful methods for solving global optimization problems. The algorithm searches an optimal point in a multi-dimensional space by adjusting the trajectories of its particles. The individual particle updates its position and velocity based on its previous best performance and previous best performance of other particles which denote *pbest* and *gbest* respectively. The position X_i^d and velocity V_i^d of the d-th dimension of i-th particle are updated as follows:

$$V_i^d \leftarrow V_i^d + c_1 \cdot rand_1 \cdot (pbest_i^d - X_i^d) + c_2 \cdot rand_2 \cdot (gbest^d - X_i^d)\,, \quad (6)$$

$$X_i^d \leftarrow X_i^d + V_i^d\,, \quad (7)$$

$pbest_i$ represents the previous best position yielding the best performance of the i-th particle; *gbest* is the best position of all particles. c_1 and c_2 denote the acceleration constants describing the weighting of each particle been pulled toward pbest and gbest respectively. $rand_1$ and $rand_2$ are two random numbers in the range [0, 1]. Let s denote the swarm size and $f()$ denote the fitness function evaluating the performance yielded by a particle.

After (6) and (7) are manipulated, the personal best position *pbest* of each particle is updated as follows

$$pbest^i = \begin{cases} pbest^i, & \text{if } f(X^i) \geq f(pbest_i), \\ X^i, & \text{if } f(X^i) < f(pbest_i), \end{cases} \quad (8)$$

and the global best position *gbest* is found by

$$gbest = \arg\min_{pbest_i} f(pbest_i), \text{ for } 1 \leq i \leq s\,. \quad (9)$$

In 2002, Clerc [12] confirms the convergence of PSO by using a constriction factor which greatly enhances the applicability of PSO. The implementation of the constriction factor is shown in (10)-(11) and this is the form used in this paper:

$$V_i^d \leftarrow \chi[V_i^d + c_1 \cdot rand_1 \cdot (pbest_i^d - X_i^d) + c_2 \cdot rand_2 \cdot (gbest^d - X_i^d)], \tag{10}$$

where

$$\chi = \frac{2}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|}, \tag{11}$$

and

$$\phi = c_1 + c_2,\ \phi > 4. \tag{12}$$

C. Q-Learning

Reinforcement learning is the problem of learning a policy from trial-and-error interactions. The learner is called the agent and a policy defines the agent's learning strategy at a given time step. In general, the objective of reinforcement learning is to maximize the expected return

$$R(t) = r(t) + \gamma \cdot r(t+1) + ... + \gamma^{\mathrm{T}} \cdot r(t+\mathrm{T}), \tag{13}$$

where γ ($0 \le \gamma \le 1$) is the discount rate and T is the terminal time step. T usually refers to the end of an trial or, in a non-episodic task, infinity. The structure of Q-learning is shown in Fig. 1. An agent applies an action $a(t)$ to the environment at a particular state $x(t)$, causing a transition from $x(t)$ to $x(t+1)$ with an immediate reward $r(t+1)$ in return. Then, the Q-function estimates the cumulative reward and is denoted by the Q-value of state-action pair ($x(t)$, $a(t)$). The Q-value of each state-action pair is updated as follows:

$$\begin{aligned} &Q(x(t),\ a(t)) \leftarrow Q(x(t),\ a(t)) + \alpha[r(t+1) + \gamma Q^*(x(t+1)) - Q(x(t),\ a(t))], \\ &Q^*(x(t+1)) = \max_{a' \in A(x(t+1))} Q(x(t+1),\ a'), \end{aligned} \tag{14}$$

where $A(x(t+1))$ is the set of candidate actions in state $x(t+1)$ and α ($0 \le \alpha \le 1$) is the learning rate.

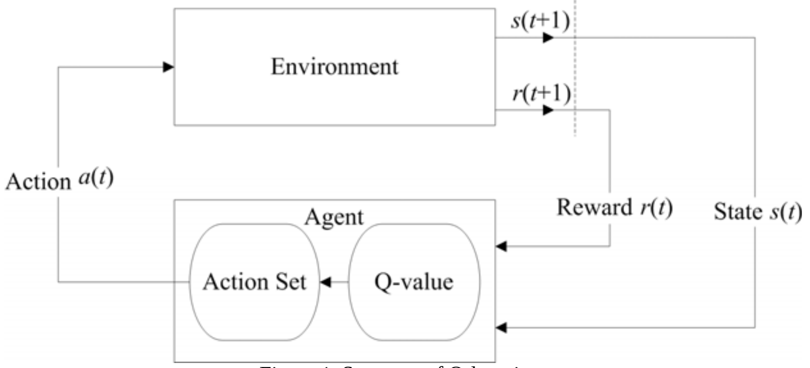


Figure 1. Structure of Q-learning.

III. LYAPUNOV DESIGN OF LEARNING AAGGENTS

Lyapunov design methods are widely used to achieve some commonly stated goals for controlling a system, such as stabilizing a system or maintaining a system's state in a desired operating range. Lyapunov function is a scalar function of the system state and it often corresponds to some real, natural notion of energy in physical systems. The basic concept of Lyapunov function has been extended to reinforcement learning problems by Perkins and Barto [23]. The purpose of [23] is to guide the system to reach and remain in a set of goal states.

The proposed QPSO conducts one Lyapunov-style theorem proposed in [23], which provides a criterion for designing the reinforcement learning agent. The theorem is listed below. Let $L : S \to \mathbb{R}$ denotes a function positive on T^c=S-T, and Δ denotes a fixed real number.

Theorem 1 *If $\forall s \notin T$, actions $a \in A(s)$, all possible next state s' (either $s' \in T$ or $L(s) - L(s') \ge \Delta$), then from any $s_t \notin T$, the environment enters T within $\lceil L(s_t)/\Delta \rceil$ time steps.*

The proof of theorem 1 can be found in [23]. Theorem 1 provides a guarantee of a plant's meeting the goal state, if the controller is designed such that it reduces the Lyapunov function of the plant in each time step. Therefore, the central task of the proposed QPSO is to identify a Lyapunov function of a control plant then design the action choices so that the reinforcement learning satisfies the above theorem. We also modify the original Q-learning algorithm into (15) so that the Q-value of a particle can indicate how soon the system's state entering the goal state:

$$Q(x(t),\ a(t)) \leftarrow Q(x(t),\ a(t)) + \alpha[-\frac{1}{t} + \gamma Q^*(x(t+1)) - Q(x(t),\ a(t))],$$
$$Q^*(x(t+1)) = \max_{a' \in A(x(t+1))} Q(x(t+1),\ a'). \tag{15}$$

IV. The Learning Algorithm of QPSO

Thorough learning algorithm of QPSO is described in this section. The architecture is shown in Fig.2. The whole learning process can be roughly divided into two parts: the Q-value and PSO operation part. The learning strategy for Q-values of particles is detailed in section IV.A while the PSO operation and the flowchart of QPSO are described in section IV.B.

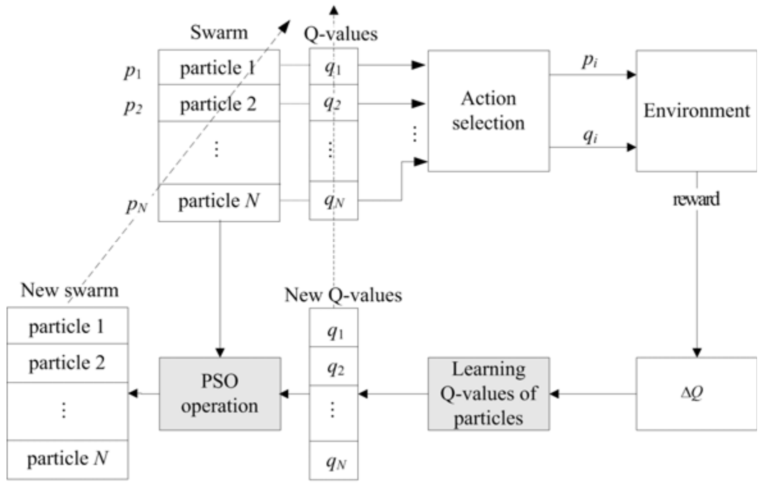


Figure 2. Architecture of QPSO

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

A. *Learning Q-values of Particles*

In QPSO learning, if there are *s* particles in the swarm, *s* trials are taken in one generation. The agent applies in each trial an action to the environment by selecting a particle based on its Q-value. Every time a particle is selected, the Q-value of the selected particle is updated based on the system's reward. As derived in (15), if the i-th particle is selected, its Q-value q_i is updated as

$$q_i(t) = q_i(t) + \alpha[-\frac{1}{t} + \gamma Q^*(x(t+1)) - q_i(t)],\ \text{for } i = 1, \cdots, s, \tag{16}$$

where

$$Q^*(x(t+1)) = \max_{a' \in A(x(t+1))} Q(x(t+1),\ a') = \max_{i=1\ldots s} Q(x(t+1),\ p_i) = \max_{i=1\ldots s} q_i(t) = q_{i^*}(t)\ ; \tag{17}$$

that is,

$$q_i(t) = q_i(t) + \alpha[-\frac{1}{t} + \gamma q_{i^*}(t) - q_i(t)] = q_i(t) + \alpha\delta_i(t),\ \text{for } i = 1, \cdots, s, \tag{18}$$

where $\delta_i(t)$ is regarded as TD error.
The new Q-values of all particles calculated from (18) are subsequently adopted as the fitness values for PSO evolution.

B. *Q-value Based PSO*

The PSO operation used in QPSO consists of two major steps: swarm initialization and Q-valued base PSO evolution. Details of these two steps are described step-by step as follows.

- Swarm initialization

The particle swarm is composed of particles encoded by the parameters on a NFS. Each particle is encoded by the mean, deviation of Gaussian membership functions and the weightings for output action strength. The number of fuzzy rules determines the length of each particle. After the number of rules is set, the initial particles are generated according to the following equations:

$$\text{Mean: } p_i[n] = random[m_{\min}, m_{\max}], \tag{19}$$

where n=1,3,⋯,2NR-1; i=1,2, ⋯,s.

$$\text{Deviation: } p_i\left[n\right] = random\left[\sigma_{\min}, \sigma_{\max}\right], \tag{20}$$

where n=2,4,⋯,2NR.

$$\text{Weight: } Chr_{g,c}\left[n\right] = random\left[w_{\min}, w_{\max}\right], \tag{21}$$

where n=2NR, 2NR+1,⋯,D.

p_i represents the *i-th* particle in the swarm; N represents the input dimension; R represents the number of fuzzy rules; D represents the size of each particle, usually D equals to $(N+1)R$ in most of cases where the dimension of output variable is 1; $[m_{min,},m_{max}]$, $[\sigma_{min,},\sigma_{max}]$ and $[w_{min,},w_{max}]$, are the predefined ranges. The above equations result in the coding scheme between a neural fuzzy system and a particle shown in Fig. 3.

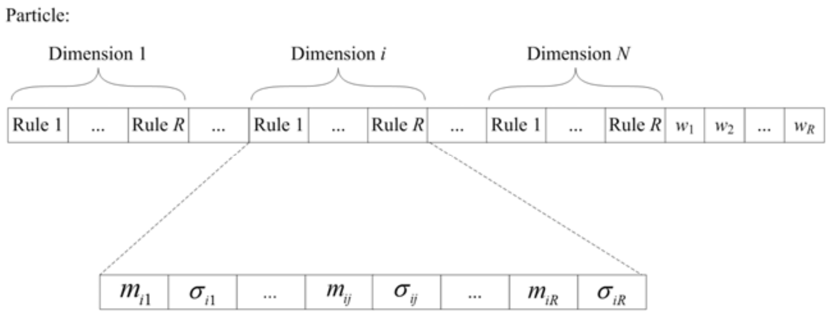


Figure 3. Coding scheme between a particle and a TSK-type NFS in QPSO.

- Q-value based PSO evolution

The Q-values derived in (18) are used as the fitness values for PSO evolution. The Q-value of each particle determines the performance of a particle for controlling the system. In the proposed QPSO, the Q-value of each particle indicates how soon a particle can guide the system's state to reach the set of goal states. The flowchart of how each particle evolves by the end of each trial is shown in Fig. 4 where *q*() represents the function to estimate a particle's Q-value and *max_gen* represents the allowance number of generations that PSO evolution runs before meeting the stopping criteria.

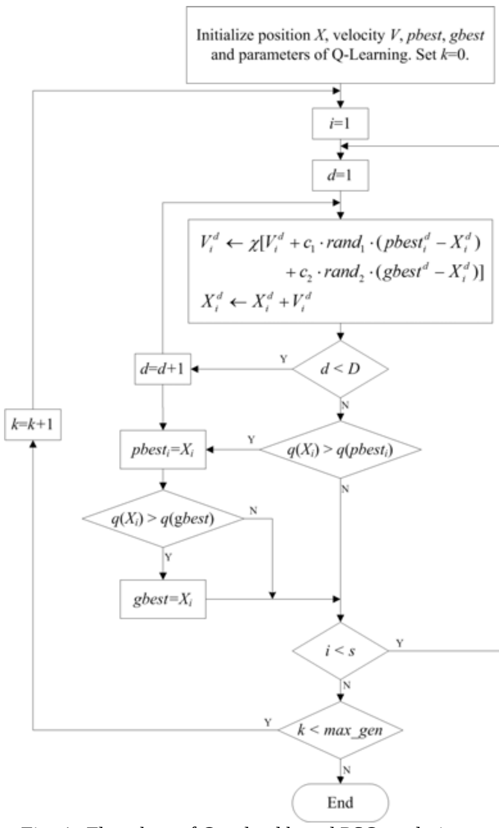

Initialize position X, velocity V, $pbest$, $gbest$ and parameters of Q-Learning. Set k=0.
i=1
d=1
$V_i^d \leftarrow \chi[V_i^d + c_1 \cdot rand_1 \cdot (pbest_i^d - X_i^d) + c_2 \cdot rand_2 \cdot (gbest^d - X_i^d)]$
$X_i^d \leftarrow X_i^d + V_i^d$
$d < D$
d=d+1
k=k+1
$q(X_i) > q(pbest_i)$
$pbest_i$=X_i
i=i+1
$q(X_i) > q(gbest)$
$gbest$=X_i
$i < s$
$k < max\_gen$
Y
N
End

Fig. 4. Flowchart of Q-valued based PSO evolution.

The learning processes of III.A and III.B proceeds to new generation until a predefined stop criterion is met. The block diagram of whole learning process in QPSO is shown in Fig. 5.

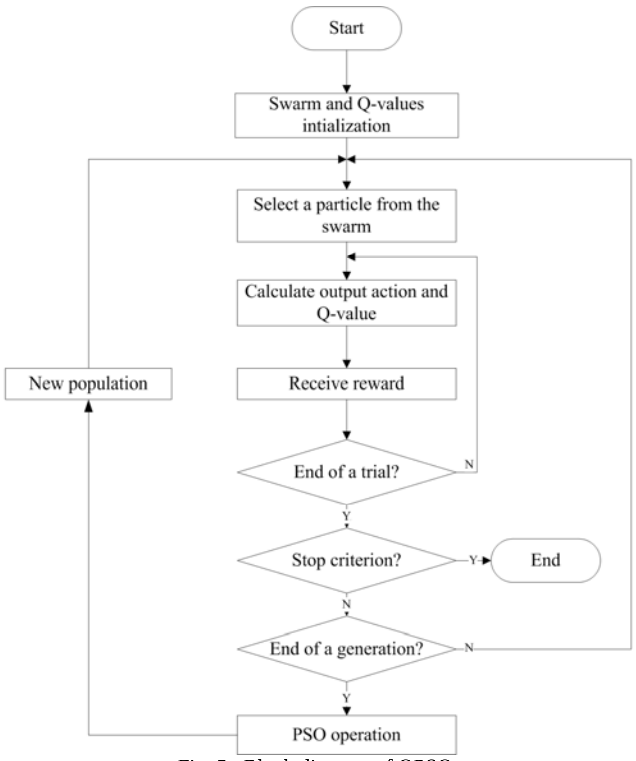


Fig. 5. Block diagram of QPSO.

V. ILLUSTRATIVE EXAMPLES

Two simulations are discussed in this section. The first simulation is the cart-pole balance control [25] and the second simulation is the control of a double-link inverted pendulum system [26].

Example 1: Control of a Single-link inverted pendulum system

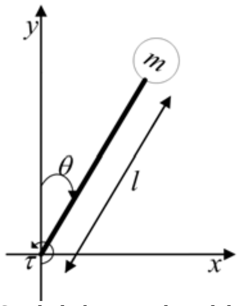


Figure 6. Single-link inverted pendulum system.

Figure 6 depicts the single-link inverted pendulum system. The state of pendulum is specified by a pole angle θ, which is measured clockwise from upright, and an angular velocity $\dot{\theta}$. The model of the single-link inverted pendulum system can be obtained as follows:

$$ml^2\ddot{\theta} = mgl\sin\theta + \tau\,, \tag{22}$$

where τ is a control torque. The mechanical energy of the pendulum including kinetic and potential is given by

$$E\left(\theta,\ \dot{\theta}\right) = \frac{1}{2}ml^2\dot{\theta}^2 + mgl\cos\theta\,. \tag{23}$$

The purpose of this control task is to determine a sequence of torques to balance the pole upright, and maintains the pole as stationary as possible. Hence, we define a goal set comprising near-upright and near-stationary states as

$$G_1 = \left\{(\theta,\dot{\theta}):\ \left\|(\theta,\dot{\theta})\right\| \le 0.01\right\}. \tag{24}$$

When the state of pendulum is in G_1, according to (23), the total mechanical energy E of the system is *mgl*. Defining a Lyapunov function $L\left(\theta,\ \dot{\theta}\right) = mgl - E\left(\theta,\ \dot{\theta}\right)$. The control objective can be also viewed as achieving $L\left(\theta,\ \dot{\theta}\right) = 0$. The time derivative of E with respect to time is given by

$$\dot{E}\left(\theta,\ \dot{\theta}\right) = \dot{\theta}\tau\,. \tag{25}$$

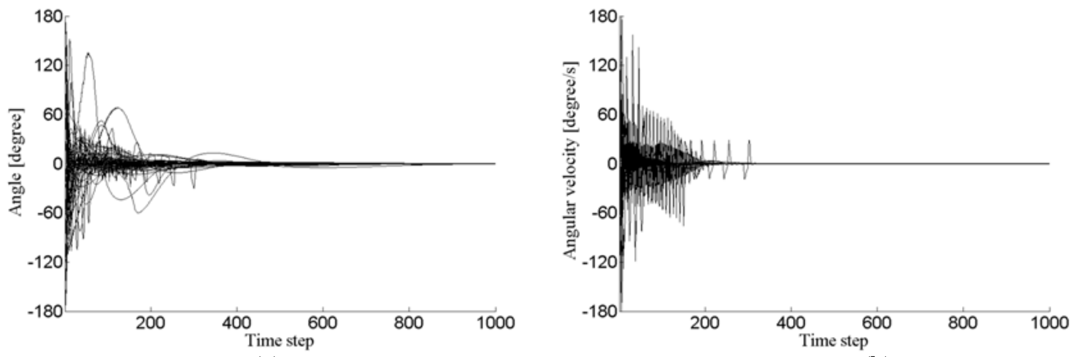
From the derivative, the energy increases if $\dot{\theta}\tau > 0$. Hence, a control law for the learning agent based on Lyapunov analysis is proposed as follows:

$$\tau = \begin{cases} sgn(\dot{\theta})z, \ \text{if}\ E\left(\theta,\ \dot{\theta}\right) < mgl, \\ 0, \ \text{if}\ E\left(\theta,\ \dot{\theta}\right) \ge mgl, \end{cases} \tag{26}$$

where $sgn(\dot{x}) = \{+1 \text{ if } \dot{x} \ge 0, \text{ and -1 if } \dot{x} < 0\}$ and z is the output of the NFS limited in [0,10]. This control law, parameterized by z, supplies a control torque of magnitude z in the direction of $\dot{\theta}$ most of the time.

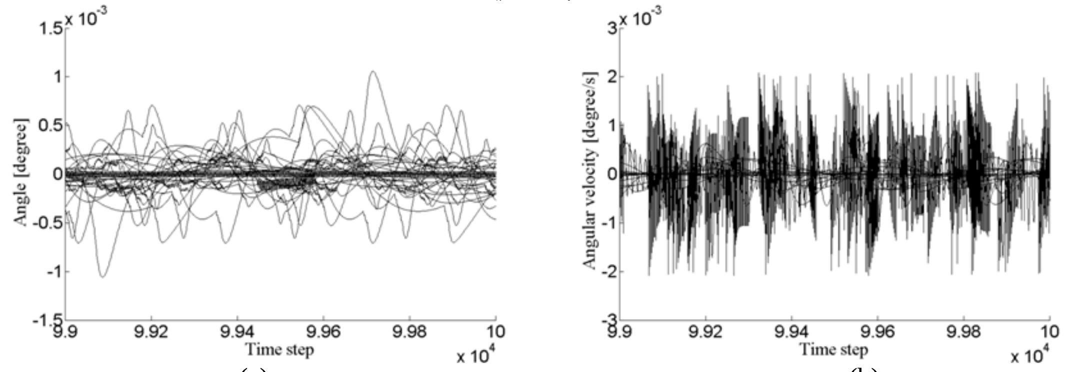
Single-link inverted pendulum system parameters are l=1m, m=1kg, g=9.8m/s. In designing the NFS, the four controller input $(\theta,\dot{\theta},x,\dot{x})$ are normalized between 0 and 1, and the output z is limited between 0 and 10. [$m_{min,}$,m_{max}], [$\sigma_{min,}$,σ_{max}] and [$w_{min,}$,w_{max}] are set as [0, 2], [0, 2] and [-30, 30], respectively. In the PSO, the swarm size s is set as 20, *max_gen* is set as 50 and the acceleration constants c_1 and c_2 are both set as 2.01. The parameters for Q-learning are set as α=0.01 and γ=0.9. Learning trials are deemed successful if they bring the system state to G_1; on the contrary, learning trials are terminated and said to be unsuccessful if they exceeded 1000 time steps. The fitness value of each particle is defined according to (18). The higher fitness value by the end of each trial represents the sooner the plant meets the goal set. When the QPSO is stopped, the best particle from the swarm in the final generation is applied to the testing phase of the single-link inverted pendulum system.

In the testing phase of this simulation, 50 runs are carried out and each run executes 100,000 time steps. The reason that the number of time steps is longer in the testing phase is to verify the ability of QPSO maintaining the environment into goal set. The testing results, which consist of the angle (degree), and the angular velocity of the pole (degree/s), are shown in Fig. 7 and 8. Each line in Fig. 7 and 8 represents a single run that starts form different initial states. Figure 15 shows the results the first 1,000 of 100,000 control time steps while Fig. 16 shows the last 1,000. As shown in Fig. 7, the QPSO successfully brings the environment of single-link inverted pendulum system to goal set G_1 in all 50 runs. From Fig. 7 we can see that with the aid of Lyapunov design, the QPSO is able to control the single-link inverted pendulum system well under different initial conditions. Trajectories shown in Fig. 8 verify the ability of the QPSO marinating the environment into G_1.



(a) (b)

Figure 7. First 1000 time steps control results of the single-link inverted pendulum system. (a) Angle of the pendulum. (b) Angular velocity of the pendulum.



(a) (b)

Figure 8. Last 1000 time steps control results of the single-link inverted pendulum system. (a) Angle of the pendulum. (b) Angular velocity of the pendulum.

To verify with the performance of the QPSO, the TD and GA based reinforcement learning (TDGAR) [20], the on-line clustering and Q-value based GA reinforcement learning (CQGAF) [21] and the recurrent wavelet-based NFS with a reinforcement group cooperation-based symbiotic evolution (R-GCSE) algorithm [22] are applied to the same control task. In the TDGAR, there are five hidden nodes and five rules in the critic network and the action network. The population size is set as 200 and the maximum perturbation is set as 0.0005. In the CQGAF, after trial-and-error tests, the final average number of rules from 50 runs was 6 by using the on-line clustering algorithm. The population size is set as 20. The parameters for Q-learning are set as α=0.01, λ=0.9 and γ=0.9. In the R-GCSE, the population size is set as 30 and the mutation rate is set as 0.01. In the QPSO and the compared approaches, a trail begins at a near upright state (5°,0, 0, 0) and ends when the control goal is met or a failure occurs. The control goal in this performance comparison test is defined as "bringing the plant to G_1 and maintaining it for 10000 time steps." A failure learning trial occurs if it exceeded 100000 time steps or the pendulum deviates beyond the range of $\pm$12°.

TABLE I. SUMMARY STATISTICS FOR EXAMPLE 1.

Methods	QPSO	TDGAR	CQGAF	R-GCSE
% of first 10% trials meeting goal.	96	56	70	78
% of trials meeting goal.	98	84	90	94
Time to goal, first 10% trials.	24.2 $\pm$ 0.8	200.2 $\pm$ 18.1	50.6 $\pm$ 7.2	78.9 $\pm$ 8.8
Average Time to goal.	21.6 $\pm$ 0.3	169.8 $\pm$ 12.9	34.2 $\pm$ 6.1	46.1 $\pm$ 4.9

The performances of all these compared methods are shown in Table I, from which we can see that the QPSO has the successful control rate and requires the least CPU-time cost. The superiority can be seen especially from the first 10% learning trials where learning agents are not fully trained yet. This is one of the best benefits of incorporating the Lyapunov design principles. The QPSO is able to apply a safe, reliable control result during initial leanings, which is crucial important in many applications.

Example 2: Control of a Double-link Inverted Pendulum System

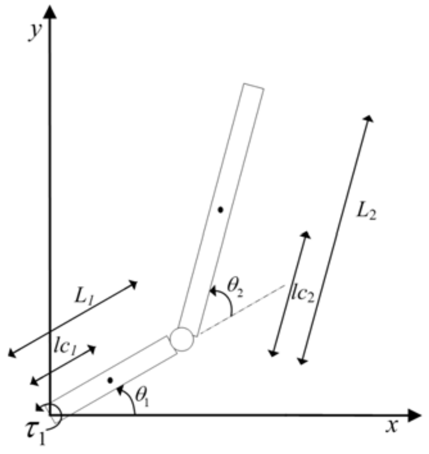


Figure 9. Double-link inverted pendulum system.

Consider the double-link inverted pendulum system: m_1 is the mass of link 1, m_2 is the mass of link 2, θ_1 is the angle that link 1 makes with the horizon, θ_2 is the angle that link 2 makes with link 1, l_1 and l_2 are the lengths of link 1 and 2, lc_1 is the distance of the center of mass of link 1, lc_2 is the distance of the center of mass of link 2, I_1 and I_2 are the moments of inertia of link 1 and link 2 about their centroids and τ_1 is the only control torque applied to the joint of link 1. We introduce the following five parameters:

$$\begin{cases} p_1 = m_1 lc_1{}^2 + m_2 l_1^2 + I_1, \\ p_2 = m_2 lc_2{}^2 + I_2, \\ p_3 = m_2 l_1 lc_1, \\ p_4 = m_1 lc_1 + m_2 l_1, \\ p_5 = m_2 lc_2, \end{cases} \tag{27}$$

the model of the system can be obtained by using Lagrange's method:

$$D(q)\ddot{q} + C(q,\ \dot{q})\dot{q} + G(q) = \tau\ , \tag{28}$$

where

$$q = [q_1,\ q_2]^T = [\theta_1,\ \theta_2]^T\ ,\quad \tau = [\tau_1,\ 0]^T\ , \tag{29}$$

$$D(q) = \begin{bmatrix} p_1 + p_2 + 2p_3\cos q_2 & p_2 + p_3\cos q_2 \\ p_2 + p_3\cos q_2 & p_2 \end{bmatrix}, \tag{30}$$

$$C(q, \dot{q}) = p_3 \sin q_2 \begin{bmatrix} -\dot{q}_2 & -\dot{q}_2 - \dot{q}_1 \\ \dot{q}_1 & 0 \end{bmatrix}, \tag{31}$$

$$G(q) = \begin{bmatrix} p_4\cos q_1 + p_5 g\cos(q_1 + q_2) \\ p_5 g\cos(q_1 + q_2) \end{bmatrix}. \tag{32}$$

The potential energy of the double-link inverted pendulum system can be defined as

$$P(q) = p_4 g\sin q_1 + p_5 g\sin(q_1 + q_2)\ , \tag{33}$$

and the total mechanical energy of the system is given by

$$\begin{aligned} E(q,\ \dot{q}) &= \frac{1}{2}\dot{q}^T D(q)\dot{q} + P(q) \\ &= \frac{1}{2}\dot{q}^T D(q)\dot{q} + p_4 g\sin q_1 + p_5 g\sin(q_1 + q_2). \end{aligned} \tag{34}$$

Therefore, from (30)-(34) we obtain that

$$\begin{aligned} \dot{E}(q,\ \dot{q}) &= \dot{q}^T D(q)\ddot{q} + \frac{1}{2}\dot{q}^T \dot{D}(q)\dot{q} + \dot{q}^T G(q) \\ &= \dot{q}^T\left(-C(q,\ \dot{q})\dot{q} - G(q) + \tau\right) + \frac{1}{2}\dot{q}^T \dot{D}(q)\dot{q} + \dot{q}^T G(q) \\ &= \dot{q}^T\tau\ = \dot{q}_1\tau_1, \end{aligned} \tag{35}$$

which shows the derivative of E is proportional to the product of the angular velocity of link 1 and the input torque.

The control objective is to stabilize the system around its top position, i.e. $(q_1, \dot{q}_1, q_2, \dot{q}_2)$ =(π/2,0,0,0). Denote $\tilde{q}_1 = q_1 - \pi / 2$, another goal set is defined by

$$G_2 = \left\{ (\tilde{q}_1, \dot{q}_1, q_2, \dot{q}_2) : \ \left\| (\tilde{q}_1, \dot{q}_1, q_2, \dot{q}_2) \right\| \le 0.01 \right\}. \tag{36}$$

When the state of double-link inverted pendulum system is in G_2, the total mechanical energy E of the system is given by

$$E(\pi/2, 0, 0, 0) = E_{\text{top}} = (p_4+p_5)g. \tag{37}$$

By defining a Lyapunov function $L(q,\ \dot{q}) = \frac{1}{2}\left[E(q,\ \dot{q}) - E_{\text{top}} \right]^2$, the control objective can be either considered as guiding the system state into G_2 or achieving $L(q,\ \dot{q}) = 0$. Hence in this paper, a control law for the learning agent in this example is proposed as follows:

$$\tau_1 = \begin{cases} sgn(\dot{q}_1)z, \text{ if } E(q,\ \dot{q}) < E_{\text{top}}, \\ 0, \text{ if } E(q,\ \dot{q}) \ge E_{\text{top}}, \end{cases} \tag{38}$$

where z is the output of the NFS limited in [0,10]. Double-link inverted pendulum system parameters are L_1=1m, L_2=2m, m_1=1kg, m_2=2kg, g=9.8m/s. In designing the NFS, the four controller input $(\theta, \dot{\theta}, x, \dot{x})$ are normalized between 0 and 1, the output z is limited between 0 and 10. [m_{min},m_{max}], [σ_{min},σ_{max}] and [w_{min},w_{max}] are set as [0,2], [0,2] and [-30,30], respectively. In the PSO, the swarm size s is set as 40, *max_gen* is set as 50 and the acceleration constants c_1 and c_2 are both set as 2.01. The parameters for Q-learning are set as α=0.01 and γ=0.9. Learning trials are deemed successful if they bring the plant to G_2; on the contrary, learning trials are terminated and said to be unsuccessful if they exceeded 1000 time steps. When the QPSO is stopped, the best particle from the swarm in the final generation is applied to the testing phase of the single-link inverted pendulum system.

In the testing phase of this simulation, 50 runs are carried out and each run executes 100,000 time steps. The testing results, which consist of the angle (degree) and the angular velocity (degree/s) of link 1 and link 2, are shown in Fig. 10 and 11. Figure 10 shows the results the first 1,000 of 100,000 control time steps while Fig. 11 shows the last 1,000. As shown in Fig. 10, the QPSO successfully brings the environment of double-link inverted pendulum system to goal set G_2 in all 50 runs under different initial environments. Trajectories shown in Fig. 11 verify the ability of the QPSO marinating the environment into G_2.

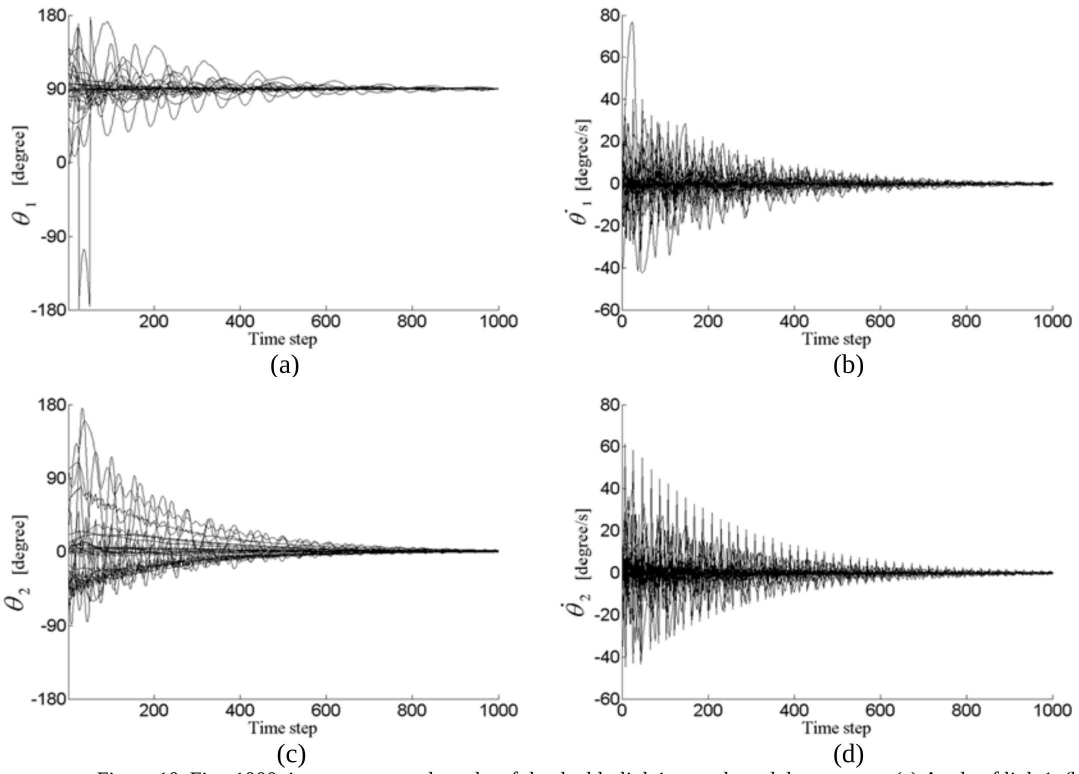


Figure 10. First 1000 time steps control results of the double-link inverted pendulum system. (a) Angle of link 1. (b) Angular velocity of link 1. (c) Angle of link 2. (d) Angular velocity of link 2.

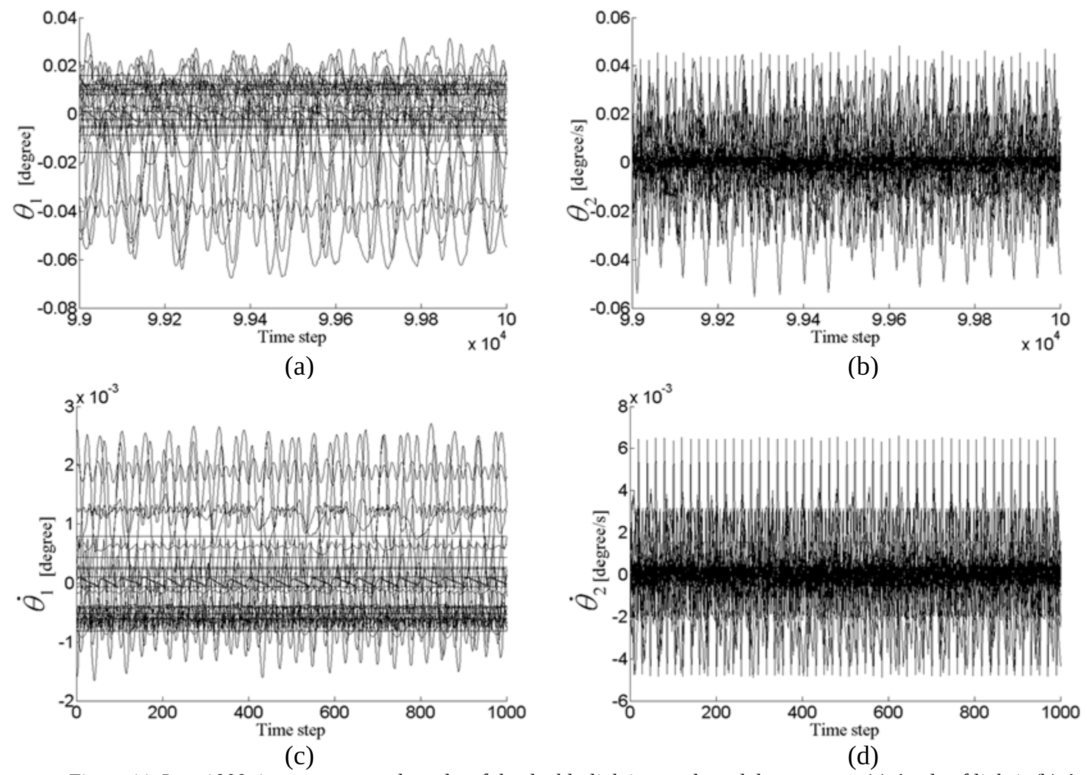


Figure 11. Last 1000 time steps control results of the double-link inverted pendulum system. (a) Angle of link 1. (b) Angular velocity of link 1. (c) Angle of link 2. (d) Angular velocity of link 2.

The TD and GA based reinforcement learning (TDGAR), the on-line clustering and Q-value based GA reinforcement learning (CQGAF) and the recurrent wavelet-based NFS with a reinforcement group cooperation-based symbiotic evolution (R-GCSE) algorithm are applied to the same control task to verify the performance of the QPSO. In the TDGAR, there are five hidden nodes and five rules in the critic network and the action network. The population size is set as 300 and the maximum perturbation is set as 0.0005. In the CQGAF, after trial-and-error tests, the final average number of rules from 50 runs was 8. The population size is set as 40. The parameters for Q-learning are set as α=0.01, λ=0.9 and γ=0.9. In the R-GCSE, the population size is set as 40 and the mutation rate is set as 0.01. In the QPSO and the compared approaches, a trail begins at a near upright state (85°, 0, 5°, 0) and ends when the control objective is met or a failure occurs. The control objective in this performance comparison test is defined as "bringing the plant to G_2 and maintaining it for 10000 time steps." A failure learning trial occurs if it exceeded 100000 time steps.

TABLE II. SUMMARY STATISTICS FOR EXAMPLE 2

Methods	QPSO	TDGAR	CQGAF	R-GCSE
% of first 10% trials meeting goal.	94	38	52	48
% of trials meeting goal.	96	56	74	84
Time to goal, first 10% trials.	36.2 ± 0.7	304.2 ± 15.1	100.7 ± 10.9	110 ± 15.7
Average Time to goal.	33.6 ± 0.4	270.8 ± 9.2	86.2 ± 8.4	90.5 ± 7.5

The performances of all these compared methods are shown in Table II. The results are similar to those from Table I which shows that, the QPSO is the most effective and efficient one among these compared methods. But one drawback of the QPSO is that it requires the most priori knowledge among these compared methods.

VI. CONCLUSION

In this paper, we propose a combination of PSO and Q-value based reinforcement learning for neuro-fuzzy system design. We have extended the PSO to work under reinforcement learning environments where only weak reinforcement signals are available. In the reinforcement learning, we use the Lyapunov design principles. Instead of training the agent how to switch control policy, we use the Lyapunov domain knowledge to derive the sign of action taken at each time step. The magnitude of actions are learned on a neuro-fuzzy system trained by PSO. With the assistance of Lyapunov design, the QPSO can enjoy several qualitative objectives. The simulations also verify the feasibility and efficiency of the proposed QPSO.

REFERENCES

[1] J. J. Buckley and Y. Hayashi, “Neural nets for fuzzy-systems,” Fuzzy Sets And System, vol. 71, no. 3, pp. 265-276, May 1995.

[2] G. G. Towell and J. W. Shavlik, “Extracting refined rules from knowledge-based neural networks,” Machine Learning, vol. 13, pp. 71-101, 1993.

[4] R. Caruana and A. Niculescu-Mizil, “An Empirical Comparison of Supervised Learning Algorithms,” Proc. Int. Conf. Machine Learning, pp. 161-168 , 2006.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning*. Cambridge, MA: MIT Press, 1998.

[6] J. N. Tsitsiklis and B. V. Roy, “An analysis of temporal-difference with function approximation,” IEEE Trans. Autom. Control, vol. 42, no. 5, pp. 834-836, Sep. 1983

[7] C. J. Wakins, “Learning from delayed rewards,” Ph.D. dissertation, King’s College, Cambridge, U.K., 1989.

[8] E. BARNARD, “Temporal-difference methods and Markov models,” IEEE Trans. on Systems Man and Cybernetics, vol. 23, Issue: 2 pp. 357-365, 1993

[9] C. J.Wakins and P. Dayan, “Technical note: Q-learning,” Machine Learning, vol. 8, pp. 279–292, 1992.

[10] J. Kennedy and R. C. Eberhart, “Particle Swarm Optimization,” in Proc. of IEEE Int. Conf. on Neural Networks, vol. 4, pp. 1942-1948, 1995.

[11] F. Bergh and A. P. Engelbrecht, “A Cooperative Approach to Particle Swarm Optimization,” IEEE Trans. on Evolutionary Computation, vol. 8, no. 3, pp. 58-73, 2002.

[12] M. Clerc and J. Kennedy, “The Particle Swarm—Explosion, Stability, and Convergence in a Multidimensional Complex Space,” IEEE Trans. on Evolutionary Computation, vol. 6, no. 1, pp. 225-239, 2004.

[13] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[14] L. J. Fogel, “Evolutionary programming in perspective: The top-down view,” in *Computational Intelligence: Imitating Life*, J. M. Zurada, R. J. Marks II, and C. Goldberg, Eds. Piscataway, NJ: IEEE Press, 1994.

[15] I. Rechenberg, “Evolution strategy,” in *Computational Intelligence: Imitating Life*, J. M. Zurada, R. J. Marks II, and C. Goldberg, Eds. Piscataway, NJ: IEEE Press, 1994.

[16] M. Conforth and Y. Meng, “Reinforcement Learning for Neural Networks using swarm Intelligence,” IEEE Swarm Intelligence Symposium, pp. 7, 2008

[17] C. J. Lin and Y. C. Hsu, “Reinforcement Hybrid Evolutionary Learning for Recurrent Wavelet-Based Neuro-Fuzzy Systems,” IEEE Trans. Fuzzy Systs., vol. 15, no. 4, 2007, pp. 729-745.

[18] G. R. Harik, F. G. Lobo, and D. E. Goldberg, “The compact genetic algorithm,” IEEE Trans. on Evolutionary Computation, vol. 3, pp. 287–297, Nov. 1999.

[19] S. Bandyopadhyay, C. A. Murthy, and S. K. Pal, “VGA-classifier: design and applications,” IEEE Trans. on Systems Man and Cybernetics, Part B, vol. 30, no. 6, pp. 890–895, 2000.

[20] C. T. Lin and C. P. Jou, “GA-based fuzzy reinforcement learning for control of a magnetic bearing system,” IEEE Trans. on Systems Man and Cybernetics, Part B, vol. 30, no. 2, pp. 276-289, 2000.

[21] C. F. Juang, “Combination of online clustering and Q-value based GA for reinforcement fuzzy system design,” IEEE Trans. on Fuzzy Systs., vol. 13, no. 3, pp. 289–302, 2005.

[22] Y.C. Hsu and S.F. Lin, “Reinforcement Group Cooperation based Symbiotic Evolution for Recurrent Wavelet-Based Neuro-Fuzzy Systems,” *Neurocomputing*, vol. 72, no. 10-12, pp. 2418-2432, 2009.

[23] Perkins, T.J. and A.G. Barto, “*Lyapunov design for safe reinforcement learning.*” Journal of Machine Learning Research, 2003. **3**(4-5): p. 803-832.

[24] Takagi, T. and M. Sugeno, “Fuzzy Identification of Systems and Its Applications to Modeling and Control,” IEEE Trans. on Systems Man and Cybernetics, 1985. **15**(1): p. 116-132.

[25] K. Furuta and M. Iwase, “Swing-up time analysis of pendulum,” *Bulletin of the Polish Academy of Sciences - Technical Sciences*, vol. 52, no. 3, pp. 153-163, 2004.

[26] C. Popescu, “Nonlinear control of underactuated horizontal double pendulum,” M.S. thesis, University of Florida Atlantic, Boca Raton, Florida, U.S., 2002.