Identification and Prediction Using Neuro-Fuzzy Networks with Symbiotic Adaptive Particle Swarm Optimization

Cheng-Jian Lin and Chun-Cheng Peng
Department of Computer Science and Information Engineering
National Chin-Yi University of Technology, Taichung County, Taiwan 411, R.O.C.
E-mail: cjlin@ncut.edu.tw, goudapeng@gmail.com

Chi-Yung Lee
Department of Computer Science and Information Engineering
Nankai University of Technology, Nantou, Taiwan 542, R.O.C.
E-mail: cylee@nkut.edu.tw

Keywords: particle swarm optimization, symbiotic evolution, neuro-fuzzy network, identification, prediction

Received: October 16, 2009

This study presents a novel symbiotic adaptive particle swarm optimization (SAPSO) for neuro-fuzzy network design. The proposed SAPSO uses symbiotic evolution and adaptive particle swarm optimization with neighborhood operator (APSO-NO) to improve the performance of the traditional PSO. In APSO-NO, we combine the neighborhood operator and the adaptive particle swarm optimization to tune the particles that are most significant. Simulation results have shown that the proposed SAPSO performs better and requires less computation time than the traditional PSO.

Povzetek: Razvita je nova metoda nevronskih mrež z uporabo roja delcev.

1 Introduction

Neuro-fuzzy networks (NFNs) have been demonstrated to be successful [1]-[9]. Two typical types of NFNs are the Mamdani-type and TSK-type models. In Mamdani-type NFNs [3]-[4], the minimum fuzzy implication is used in fuzzy reasoning. In TSK-type NFNs [5]-[8], the consequent of each rule is a function input variable. The generally adopted function is a linear combination of input variables plus a constant term. Many researchers [6]-[7] have shown that using TSK-type NFNs achieve superior performance in network size and learning accuracy than using Mamdani-type NFNs.

Training parameters is a problem in the design of a NFN. To solve this problem, back-propagation (BP) training is widely used [3]-[8]. It is a powerful training technique that can be applied to networks. Nevertheless, the steepest descent technique is used in BP training to minimize the error function. The algorithm may allow the local minima to be reached very quickly, yet the global solution may never be found. In addition, the performance of BP training depends on the initial values of the system parameter, and for different network topologies, new mathematical expressions for each network layer have to be derived. Considering the disadvantages mentioned above, one might be faced with suboptimal performances, even for a suitable NFN topology. Hence, techniques capable of training network parameters and finding a global solution while optimizing the overall structure are needed.

In this respect, a new algorithm, called particle swarm optimization (PSO), appears to be a better algorithm than the BP algorithm. It is an evolutionary computation technique developed by Kennedy and Eberhart in 1995 [10]. The underlying motivation for the development of the PSO algorithm was the social behavior of animals, such as birds flocking together, fish swimming in schools, and insects swarming together. Several researchers have used the PSO method to solve some optimization problems, like control problems [11]-[13] and neural network design [14]-[15].

The performance of most stochastic optimization algorithms, including the PSO and genetic algorithms (GAs), declines as the dimensionality of the search space increases. These algorithms stop when they generate a solution that falls in the optimal region, a small volume of the search space surrounding the global optimum. The probability in stochastic optimization algorithms decreases exponentially as the dimensionality of the search space increases. It is clear that, in a similar topology, it is harder to find the global optimum in a high-dimensional problem than it is in a low-dimensional problem. One way to overcome this exponential increase in difficulty is to partition the search space into lower dimensional subspaces, as long as the optimization algorithm can guarantee that it will be able to search every possible region of the search space.

In this paper, a novel learning algorithm, called symbiotic adaptive particle swarm optimization (SAPSO), that tunes the parameters of NFNs is proposed. The proposed SAPSO is different from the traditional PSO. In the traditional PSO, each particle represents a fuzzy system. But in SAPSO, each particle represents only one fuzzy rule. A R-rule fuzzy system is constructed by

selecting and combining R particles from a given swarm. The proposed SAPSO consists of symbiotic evolution and adaptive particle swarm optimization with neighborhood operator to improve the performance of the traditional PSO.

The advantages of the proposed SAPSO are summarized as follows: (1) SAPSO can reduce population sizes; (2) the computation request of SAPSO is less than that of the traditional PSO in each generation; (3) in the learning process, the relative parameters in a fuzzy system are searched; they prevent interference from other parameters that can find what the best parameter values are; (4) the adjusting parameter strategy of SAPSO is more significant than the traditional PSO.

The rest of this paper is organized as follows. After reviewing of training algorithms for NFNs in Section 2, Section 3 illustrates the structure of the TSK-type fuzzy model. An overview of PSO is given in Section 4. A novel symbiotic adaptive particle swarm optimization (SAPSO) is proposed in Section 5. Sections 6 and 7 respectively present the simulation results and discussion. Finally, the conclusion is given in the last section.

2 Related works

Besides the most-applied BP algorithm, some other traditional optimization approaches had been applied to training NFNs, such as the Broyden-Fletcher- Goldfarb-Shanno (BFGS) [16]-[17], conjugate gradient (CG) [18]-[19], and Levenberg-Marquardt (LM) [20]-[21] methods.

In the context of deterministic unconstrained optimisation, quasi-Newton (QN) methods, sometimes called variable metric methods, are well-known algorithms for finding local minima of specific functions. QN methods are based on Newton's method to find the stationary point of a function, where the gradient is zero. Newton's method assumes that the function can be locally approximated as a quadratic in the region around the optimum, and requires the first and second derivatives [22], i.e. the gradient vector and the Hessian matrix, to find the stationary point. Moreover, the Newton's method and its variants require that the Hessian is positive definite - a condition that is difficult to guarantee in practice.

Conjugate Gradient methods are in principle approaches suitable for large-scale problems [23]. The basic idea of CG methods is to find the stepsize along a linear combination of the current gradient vector and the previous search direction. On the other hand, equipped with a damping factor, the LM (so-called *damped Gauss-Newton*) methods are capable of relaxing the difficulties of Hessian-based training, i.e. the ill-conditioning of the Hessian matrix. In addition, when the damping factor is zero, the LM methods become identical to the Gauss-Newton approach; while as the damping factor gets close to infinity, the LM methods are then get equivalent to the steepest descent method.

As indicating in the Introduction section, although the traditional second-order approaches generally have faster convergent speeds, they are still in the situation of local optimization. Evolutional approaches such as particle swarm optimization (PSO) [10], differential evolution (DE) [21], and symbiotic evolution (SE) [25] have been developed for training NFNs [26]-[28], respectively. Since this paper focuses on the PSO approach, the concepts of DE and SE methods are omitted here and suggested to refer the relevant literature for further details, whereas the overview of the PSO and our proposed symbiotic adaptive PSO are presented in this paper.

3 Structure of a TSK-type neuro-fuzzy network (TNFN)

A fuzzy model is a knowledge-based system characterized by a set of rules that model the relationship between the control input and output. The reasoning process is defined by means of the inference method, aggregation operators, and fuzzy connectives. The fuzzy knowledge base contains the definitions of fuzzy sets, which are stored in a fuzzy database, and a collection of fuzzy rules.

Fuzzy rules are defined by their antecedents and consequents, which relate an observed input state to a desired control action. Most fuzzy systems employ the inference method proposed by Mamdani, in which the consequent parts are defined by fuzzy sets [1]. A Mamdani-type fuzzy rule has the form:

$$\textit{IF } x_1 \textit{ is } A_{1j}(m_{1j}, \sigma_{1j}) \textit{ and } x_2 \textit{ is } A_{2j}(m_{2j}, \sigma_{2j}) \ldots \textit{and } x_n \textit{ is } A_{nj}(m_{nj}, \sigma_{nj})$$
$$\textit{THEN } y' \textit{ is } B_j(m_j, \sigma_j) \quad (1)$$

where m_{ij} and σ_{ij} represent a Gaussian membership function with mean and deviation, respectively, of the *i*th dimension and the *j*th rule node. The consequents B*j* of the *j*th rule are aggregated into one fuzzy set for the output variable y'. Crisp action is obtained through defuzzification, which calculates the centroid of the output fuzzy set. The more common fuzzy inference method proposed by Mamdani, Takagi, Sugeno, and Kang introduced a modified inference scheme [5]. The first two parts of the fuzzy inference process, fuzzifying the inputs and applying the fuzzy operator, are exactly the same. A Takagi-Sugeno-Kang (TSK)-type fuzzy model employs different implications and aggregation methods than the standard Mamdani controller. Instead of fuzzy sets being used, the conclusion part of a rule is a linear combination of the crisp inputs.

$$\textit{IF } x_1 \textit{ is } A_{1j}(m_{1j}, \sigma_{1j}) \textit{ and } x_2 \textit{ is } A_{2j}(m_{2j}, \sigma_{2j}) \ldots \textit{and } x_n \textit{ is } A_{nj}(m_{nj}, \sigma_{nj})$$
$$\textit{THEN } y' = w_{0j} + w_{1j}x_1 + \ldots + w_{nj}x_n \quad (2)$$

Since the consequent of a rule is crisp, the defuzzification step becomes obsolete in the TSK inference scheme. Instead, the control output is computed as the weighted average of the crisp rule outputs. This computation is less expensive than calculating the center of gravity.

The structure of the TSK-type neuro-fuzzy network (TNFN) is shown in Fig. 1, where *n* and *R* are, respectively, the number of input dimensions and the number of rules. It is a five-layer network structure. The functions of the nodes in each layer are described as follows:

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

$$u_i^{(1)} = x_i, \quad i = 1 \cdots n \tag{3}$$

Layer 2 (*Membership Function Nodes*): Nodes in this layer correspond to one linguistic label of the input variables in layer 1; that is, the membership value specifying the degree to which an input value belongs to a fuzzy set is calculated in this layer. For an external input x_i, the following Gaussian membership function is used:

$$u_{ij}^{(2)} = \exp\left(-\frac{\left[u_i^{(1)} - m_{ij}\right]^2}{\sigma_{ij}^2} \right), \quad j = 1 \cdots R \tag{4}$$

where m_{ij} and σ_{ij} are, respectively, the center and the width of the Gaussian membership function of the *j*th term of the *i*th input variable x_i.

Layer 3 (*Rule Nodes*): The output of each node in this layer is determined by the fuzzy AND operation. Here, the product operation is utilized to determine the firing strength of each rule. The function of each rule is

$$u_j^{(3)} = \prod_i u_{ij}^{(2)} \tag{5}$$

Layer 4 (*Consequent Nodes*): Nodes in this layer are called consequent nodes. The input to a node in layer 4 is the output delivered from layer 3, and the other inputs are the input variables from layer 1, as depicted in Fig. 1. For this kind of node, we have

$$u_j^{(4)} = u_j^{(3)} (w_{0j} + \sum_{i=1}^{n} w_{ij} x_i) \tag{6}$$

where the summation is over all the inputs, and w_{ij} are the corresponding parameters of the consequent part. w_{ij} can be any real value. If w_{ij}=0, i >0, the TNFN model in this case will be called the zero-order TNFN model.

Layer 5 (*Output Node*): Each node in this layer corresponds to one output variable. The node integrates all the actions recommended by layers 3 and 4 and acts as a defuzzifier with

$$y = u^{(5)} = \frac{\sum_{j=1}^{R} u_j^{(4)}}{\sum_{j=1}^{R} u_j^{(3)}} = \frac{\sum_{j=1}^{R} u_j^{(3)} (w_{0j} + \sum_{i=1}^{n} w_{ij} x_i)}{\sum_{j=1}^{R} u_j^{(3)}} \tag{7}$$

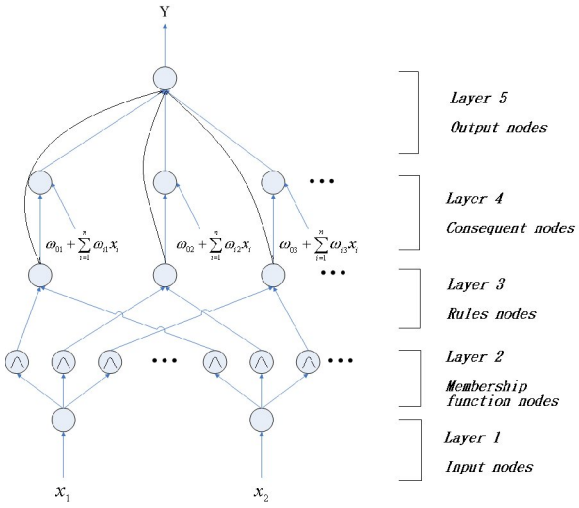


Figure 1: Structure of the TNFN model.

4 An overview of particle swarm optimization

Particle swarm optimization (PSO) [10] is a recently invented high performance optimizer that possesses several highly desirable attributes, including a basic algorithm that is easy to understand and implement. This algorithm is similar to genetic algorithms and evolutionary algorithms, but requires less computational memory and fewer lines of code. The PSO conducts searches using a population of particles which correspond to individuals in GA. Each particle has a velocity vector $\vec{v}_i$ and a position vector $\vec{x}_i$ to represent a possible solution.

Consider an optimization problem that requires the simultaneous optimization of variables. A collection or swarm of particles are defined, where each particle is assigned a random position in the *N*-dimensional problem space so that each particle's position corresponds to a possible solution of the optimization problem. The particles fly rapidly, moving at their own respective velocities, and search the space. PSO has a simple rule, namely, that each particle has three choices in evolution: (1) insist on itself; (2) move towards its own current best position; each particle remembers its own personal best position that it has found, called the local best; (3) move towards the current best position of the population; each particle also knows the best position found by any other particle in the swarm, called the global best. PSO reaches a balance among these three choices.

At each time step, each of the particle positions is scored to obtain a fitness value based on how well it solves the current problem. Using the local best position (*Lbest*) and the global best position (*Gbest*), a new velocity for each particle is updated using

$$\vec{v}_i(k+1) = \omega * \vec{v}_i(k) + \phi_1 * rand() * (Lbest - \vec{x}_i(k)) + \phi_2 * rand() * (Gbest - \vec{x}_i(k)) \quad (8)$$

where ω, ϕ_1, and ϕ_2 are called the coefficient of inertia, the cognitive study, and the society study, respectively. The term *rand()* refers to uniformly distributed random numbers in [0, 1]. The term $\vec{v}_i$ is limited to the range $\pm \vec{v}_{\max}$. If the velocity violates this limit, it will be set to its proper limit. The concept of the updated velocity is illustrated in Fig. 2.

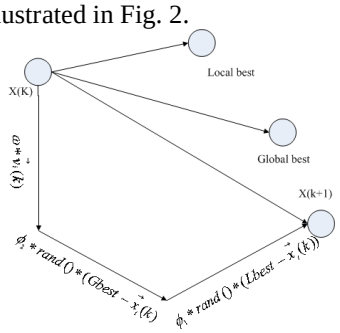


Figure 2: The diagram of the updated velocity in the PSO.

A variable velocity enables every particle to search around its individual best position and the global best position. Based on the updated velocities, each particle changes its position according to the following:

$$\vec{x}_i(k+1) = \vec{x}_i(k) + \vec{v}_i(k+1) \quad (9)$$

When every particle is updated, the fitness value of each particle is calculated again. If the fitness value of the new particle is higher than those of the local best/global best, then the local best/global best will be replaced with the new particle. When the above updating processes are repeated step-by-step, the whole population will evolve toward the optimum solution. A detailed flowchart is shown in Fig. 3.

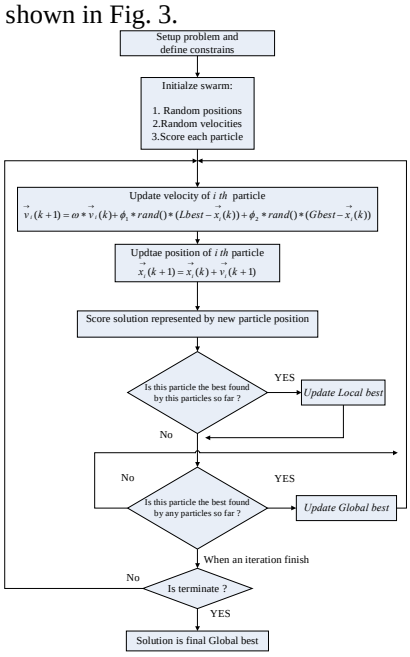


Figure 3: The typical PSO flowchart illustrates the steps and update equations.

5 The symbiotic adaptive particle swarm optimization (SAPSO)

In this section, we will introduce the symbiotic adaptive particle swarm optimization (SAPSO) for NFN design. SAPSO uses symbiotic evolution and adaptive particle swarm optimization with neighborhood operator. The detailed process is described below.

5.1 The Design of Neuro-Fuzzy Network Using SAPSO

Symbiotic evolution was first proposed in an implicit fitness-sharing algorithm that was used in an immune system model [31]. Unlike the traditional PSO that uses each particle in a swarm as a full solution to a problem, in symbiotic evolution, each individual in a population represents only a partial solution to a problem. The goal of each individual is to form a partial solution that will be combined with other partial solutions currently in the population to build an effective full solution. In a normal evolution algorithm, a single individual is responsible for the overall performance, and is assigned a fitness value according to its performance. In symbiotic evolution, the fitness of an individual (a partial solution) is calculated by summing up the fitness values of all possible combinations of that individual with other current individuals (partial solutions) and then dividing the sum by the total number of combinations. The representation of a fuzzy system using SAPSO is shown in Fig. 4.

In Fig. 4, we can see that if we need R rules to construct a fuzzy system, we will have R sub-swarms. Each sub-swarm produces its own sub-particles. The current best parameters, called the cooperative best (Cbest), of the fuzzy system are recorded. As with the traditional PSO, the velocities and sub-particles in every sub-swarm need to be updated.

The evolution process of SAPSO includes coding,

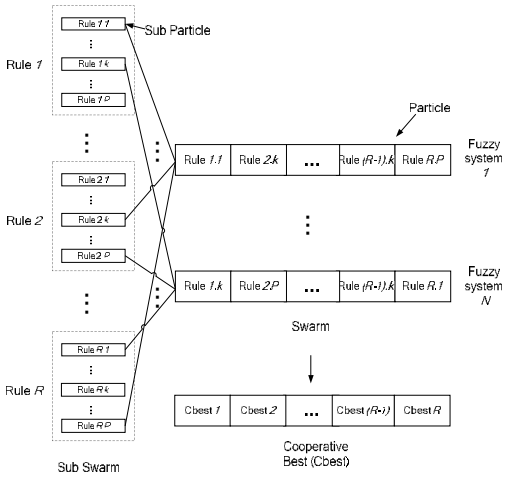


Figure 4: The representation of a fuzzy system by SAPSO.

m_{1j}	σ_{1j}	m_{2j}	σ_{2j}	…	m_{nj}	σ_{nj}	…	w_{0j}	w_{1j}	…	w_{nj}

Figure 5: Coding a rule of SAPSO into a sub-particle.

initialization, fitness assignment, and sub-particle updating. The coding step is concerned with the membership functions and the fuzzy rules of a fuzzy system that represent the particles in SAPSO. The initialization step assigns the sub-swarm values before the evolution process. The fitness assignment step gives a suitable fitness value to each fuzzy system during the evolution process. The complete learning process is described step by step below.

A. Coding step: The first step in SAPSO is to code a fuzzy rule into a sub-particle. Figure 5 shows a fuzzy rule that is given by Eq. (2), where m_{ij} and σ_{ij} represent a Gaussian membership function with mean and deviation in the *i*th dimension and the *j*th rule node, respectively.

B. Initialization step: Before SAPSO is designed, an initial sub-swarm should be generated. As in the traditional PSO, an initial sub-swarm is generated randomly within a fixed range.

C. Fitness assignment step: As mentioned before for SAPSO, the fitness value of a rule (a sub-particle) is calculated by summing up the fitness values of all the possible combinations, which are randomly selected, and then dividing the sum by the total number of combinations. The details for assigning the fitness value are described step-by-step as follows.

Step 1: Randomly choose one sub-particle from each sub-swarm and assemble it to form a particle. This particle represents a fuzzy system derived from these sub-particles.

Step 2: Evaluate the performance of every fuzzy system that is generated from Step 1 to obtain a fitness value.

Step 3: The fitness records are initially set to zero. Accumulate the fitness value of the selected sub-particles to the fitness records.

Step 4: Repeat the above steps until each rule (sub-particle) in a sub-swarm has been selected a sufficient number of times, and record how many times each sub-particle has participated in the fuzzy systems.

Step 5: Divide the accumulated fitness value of each sub-particle by the number of times it has been selected. The average fitness value represents the performance of a rule. In this paper, the fitness value is designed according to the follow formulation:

$$\text{Fitness Value} = 1/(1+\sqrt{E(y,\bar{y})/T}) \tag{10}$$

$$\text{where } E(y,\bar{y}) = \sum_{i=1}^{T} (y_i - \bar{y}_i)^2 \tag{11}$$

where y_i represents the true value of the *i*th output, $\bar{y}_i$ represents the predicted value, $E(y,\bar{y})$ is an error function, and *T* represents the number of the training data of each generation.

D. Updating velocities and sub-particles: When the fitness value of each sub-particle is obtained from the fitness assignment step, the *Lbest* of each sub-particle and the *Gbest* of each sub-swarm are updated simultaneously using adaptive particle swarm optimization with neighborhood operator (APSO-NO). The algorithm of APSO-NO is described in subsection 4.2.

E. Updating cooperative best (Cbest): When the fitness value of every fuzzy system is obtained, we can find the best fuzzy system in each generation. If the fitness value of any fuzzy system is higher than the best cooperative one, the cooperative best will be replaced.

The steps mentioned above are repeated until the predetermined condition is achieved.

5.2 The Adaptive Particle Swarm Optimization with Neighborhood Operator (APSO-NO)

In recent years, many researchers [30], [32] have proposed using stability analysis on dynamic PSO in order to obtain an understanding on how it searches for a global optimal solution and the strategy it uses to tune parameters.

In this paper, the velocity of a particle at the (*k*+*1*)-th iteration is redefined for SAPSO as follows:

$$\begin{aligned}\vec{v}_i(k+1) = {} & \omega * \vec{v}_i(k) + \phi_1 * rand() * (Lbest - \vec{x}_i(k)) \\ & + \phi_2 * rand() * (Gbest - \vec{x}_i(k)) \\ & + \phi_3 * rand() * (Cbest - \vec{x}_i(k))\end{aligned} \tag{12}$$

where ω, ϕ_1, ϕ_2, and ϕ_3 are called the coefficient of inertia, cognitive study, group study, and society study, respectively. We hope to accelerate every sub-particle in a direction toward the best self (Gbest), the best of a partial solution (Lbest), and the best of the full solution (Cbest). The particle will be reduced to one dimension for easy analysis. Thus, Eq. (12) is rewritten as follows:

$$v(k+1) = \omega * v(k) + \alpha * (Z - x(k)) \tag{13}$$

$$\text{where } \alpha = \phi_1 * rand() + \phi_2 * rand() + \phi_3 * rand() \tag{14}$$

$$Z = \frac{\phi_1 * rand() * Lbest + \phi_2 * rand() * Gbest + \phi_3 * rand() * Cbest}{\alpha} \tag{15}$$

The reduced formulas of APSO-NO can be expressed as follows:

$$\begin{aligned} & v(k+1) = \omega * v(k) + \alpha * y(k) \\ & y(k+1) = -\omega * v(k) + (1-\alpha) * y(k) \\ & where \ \ y(k) = Z - x(k) \end{aligned} \tag{16}$$

The reduced system can then be written into a form using matrix algebra.

$$P_{k+1} = MP_k \tag{17}$$

$$where\ P_k = \begin{bmatrix} v(k) \\ y(k) \end{bmatrix},\ M = \begin{bmatrix} \omega & \alpha \\ -\omega & 1-\alpha \end{bmatrix}$$

More generally, $P_k = M_k P_0$. Thus, the system is defined completely by *M*. The eigenvalues of *M* are

$$\begin{cases} \lambda_1 = \dfrac{\omega+1}{2} - \dfrac{\alpha}{2} + \dfrac{\sqrt{(\omega+1-\alpha)^2 - 4\omega}}{2} \\ \lambda_2 = \dfrac{\omega+1}{2} - \dfrac{\alpha}{2} - \dfrac{\sqrt{(\omega+1-\alpha)^2 - 4\omega}}{2} \end{cases} \quad (18)$$

According to stability theory, the behavior of a particle is stable if and only if $|\lambda_1|<1$ and $|\lambda_2|<1$. Since the eigenvalues λ_1, λ_2 are a function of the parameters ω, ϕ_1, ϕ_2, and ϕ_3, eigenvalue analysis will be carried out under the following four conditions to find the stable condition of the system. The detailed proofs refer to [30].

$$\begin{cases} (1)\ \omega = 0, & \Rightarrow 0 < \alpha < 2 \\ (2)\ \alpha < \omega + 1 - 2\sqrt{\omega}, & \Rightarrow 0 \le \omega < 1 \\ (3)\ \omega + 1 - 2\sqrt{\omega} < \alpha < \omega + 1 + 2\sqrt{\omega}, & \Rightarrow 0 \le \omega < 1 \\ (4)\ \omega + 1 + 2\sqrt{\omega} < \alpha, & \Rightarrow 0 \le \omega < 1 \end{cases} \quad (19)$$

Based on the above analysis and with the use of the parameters α and ω, the criterion of convergence $|\lambda_1|<1$ and $|\lambda_2|<1$ can be written as follows:

$$\begin{aligned} &0 < \alpha < 2\omega + 2 \\ &0 \le \omega < 1 \end{aligned} \quad (20)$$

The analysis of vibration for a period is also investigated in [30]. In condition (3) of Eq. (29), since $(\omega+1\text{-}\alpha)^2 - 4\omega < 0$ when $\omega + 1 - 2\sqrt{\omega} < \alpha < \omega + 1 + 2\sqrt{\omega}$, λ becomes a complex number, and α can be described as follows:

$$\begin{aligned} &\alpha = \omega + 1 - 2\sqrt{\omega} + 4\delta\sqrt{\omega} \\ &where\ 0 < \delta < 1 \end{aligned} \quad (21)$$

Because λ is a complex number, it can be expressed as a polar coordinate $\lambda = |\lambda| e^{\theta}$. If T is to be the period of the reduced system,

$$\begin{aligned} T = \frac{2\pi}{\theta} &= \frac{2\pi}{\tan^{-1}\dfrac{\mathrm{Im}}{\mathrm{Re}}} \\ &= \frac{2\pi}{\tan^{-1}\dfrac{\dfrac{\sqrt{-(\omega+1-\alpha)^2 + 4\omega}}{2}}{\dfrac{\omega+1-\alpha}{2}}}, \quad \alpha < \omega + 1 \end{aligned} \quad (22)$$

If Eq. (21) is used instead of α, then Eq. (22) can be written as follows:

$$\begin{aligned} T &= \frac{2\pi}{\tan^{-1}\dfrac{2\sqrt{\delta-\delta^2}}{1-2\delta}} \qquad (\delta < 0.5) \\ &= \frac{2\pi}{\pi + \tan^{-1}\dfrac{2\sqrt{\delta-\delta^2}}{1-2\delta}} \qquad (\delta > 0.5) \end{aligned} \quad (23)$$

It was shown from Eq. (14) that is a random number distributed in [0, $(\phi_1+\phi_2+\phi_3)$)] and that its average value is $\frac{\phi_1+\phi_2+\phi_3}{3}$. We use three parameters, κ_1, κ_2 and κ_3, where $0 < \kappa_1 + \kappa_2 + \kappa_3 < 1$. Therefore, ϕ_1, ϕ_2, and ϕ_3 is

$$\phi_1 : \phi_2 : \phi_3 = \kappa_1 : \kappa_2 : \kappa_3 \Rightarrow \begin{cases} \phi_1 = 3\kappa_1\alpha \\ \phi_2 = 3\kappa_2\alpha \\ \phi_3 = 3\kappa_3\alpha \end{cases} \quad (24)$$

We can redefine Eq. (12) based on the above results as follows:

$$\begin{aligned} \vec{v}_i(k+1) = \omega * \vec{v}_i(k) &+ 3\kappa_1\alpha * rand() * (Lbest - \vec{x}_i(k)) \\ &+ 3\kappa_2\alpha * rand() * (Gbest - \vec{x}_i(k)) \\ &+ 3\kappa_3\alpha * rand() * (Cbest - \vec{x}_i(k)) \end{aligned} \quad (25)$$

where $\alpha = \omega + 1 - 2\sqrt{\omega} + 4\delta\sqrt{\omega}$ (26)

The qualitative relation between the search trajectory of the reduced system and the parameters is summarized as follows:

(1).If $0 \le \omega < 1$ and ω closes to 0, the convergence tendency of the system will increase. If $1 < \omega$ and ω faces to ∞, the divergence tendency of the system will increase.

(2).If $0 < \alpha < 1$ and α closes to 1, the dynamics of the system becomes a vibration.

(3). If the parameters $0 < \kappa_1 + \kappa_2 + \kappa_3 < 1$ and any of the other coefficients closes to 1, Z moves toward the corresponding best (*Lbest, Gbest or Cbest*).

In addition to the above method, another concept, called the neighborhood operator, is introduced in [15]. It prevents a swarm from tending to the global best prematurely in the search process. If it commits early to the global best in the search process, it may be trapped in a poor local minimum. In the neighborhood operator, the Gbest is not a fixed value. Therefore, Eq. (25) can be rewritten as follows:

$$\begin{aligned} \vec{v}_i(k+1) = \omega * \vec{v}_i(k) &+ 3\kappa_1\alpha * rand() * (Lbest - \vec{x}_i(k)) \\ &+ 3\kappa_2\alpha * rand() * (Vbest - \vec{x}_i(k)) \\ &+ 3\kappa_3\alpha * rand() * (Cbest - \vec{x}_i(k)) \end{aligned} \quad (27)$$

where *Vbest* is defined as the best solution in the neighborhood around the sub-particles that are waiting to be updated. The neighborhood is identified by calculating the distances between the candidate sub-particles and the other sub-particles. The pseudo code is shown in Fig. 6. We can see that the number of neighborhoods gradually increases according to the generations. When the generations are close to terminal conditions, *Vbest* tends toward *Gbest*.

1. Get *dist[i]* by calculating distances between the candidate sub-particle and all other sub-particle.
2. Find the $dist_{\max}$ from *dist[i]*.
3. Define a threshold $\ell = 0.5 + 0.5 * (iteration_{now} / iteration_{\max})$
4. if ℓ <0.8

   if $\ell >$ *dist[i]*/ $dist_{\max}$

Figure 6: The pseudo code for finding Vbest in every iteration.

6 Simulation results

In this section, the proposed SAPSO is applied to the TNFN design and compared with the traditional PSO. Both SAPSO and the traditional PSO are used to adjust

the antecedent and consequent parameters of fuzzy rules in TNFN.

We use three different simulations for all methods. The first simulation uses the example given by Narendra and Parthasarathy [33]. The second simulation predicts the chaotic time series [34], and the third example approximates a piecewise function [35]. In our simulations, the numbers of swarms and sub-swarms are set to 50 and 10. The initial parameters of the traditional PSO and SAPSO are given in Table 1. In SAPSO, we use different parameter values to observe the effect on performance. All the programs are developed using MATLAB 6.1 software, and each problem is simulated on a Pentium III 1GHz desktop computer. Each experiment is run 20 times.

Parameter / Model	ω	$\kappa_1(\phi_1)$	$\kappa_2(\phi_2)$	κ_3	δ
Traditional PSO	0.4	2.0	2.0	NA	NA
SAPSO*1*	0.4	0.3	0.3	0.3	0.5
SAPSO*2*	0.4	0.5	0.5	0	0.5
SAPSO*3*	0.4	0.5	0	0.5	0.5
SAPSO*4*	0.4	0.2	0.4	0.4	0.5
SAPSO*5*	0.4	0.2	0.4	0.4	0.6-0.4
SAPSO*6*	0.4	0.1	0.3	0.6	0.3
SAPSO*7*	0.4	0.1	0.3	0.6	0.5
SAPSO*8*	0.4	0.1	0.3	0.6	0.7
SAPSO*9*	0.4	0.1	0.3	0.6	0.6-0.4

Table 1: The initial parameters of the traditional PSO and the SAPSO.

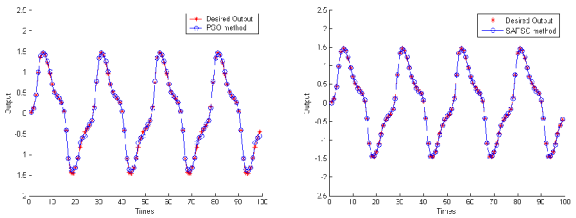
Example 1-Identification of Nonlinear Dynamic System

The first example used for identification is described by the difference equation

$$y(k+1)=\frac{y(k)}{1+y^2(k)}+u^3(k) \qquad (28)$$

The output of this equation depends nonlinearly on both its past value and the input, but the effects of the input and output values are not additive. The training input patterns are randomly generated in the interval [-2, 2] for the training data. In this problem, we use five fuzzy rules, and evolution progressed for 1000 generations.

After 1000 generations, the average best root mean square error (RMSE) of the output approximates 0.016. Figures 7 (a)-(b) show the outputs of the two methods for the input $u(k)=sin(2\pi k/25)$. Figure 8 and Table 2 show the learning curves and the performance of PSO and SAPSO with different parameter values.



(a) (b)

Figure 7: Results of the desired output and the model output of (a) the PSO method, and (b) the SAPSO method.

RMSE / Model	RMSE(Ave)	RMSE(Best)
Traditional PSO	0.023	0.011
SAPSO*1*	0.100	0.059
SAPSO*2*	0.068	0.024
SAPSO*3*	0.066	0.035
SAPSO*4*	0.031	0.012
SAPSO*5*	0.038	0.024
SAPSO*6*	0.099	0.057
SAPSO*7*	0.037	0.020
SAPSO*8*	0.119	0.108
SAPSO*9*	**0.016**	**0.012**

Table 2: The performance comparison with two methods.

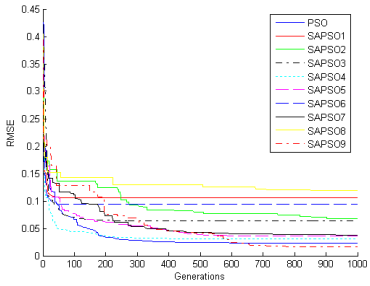


Figure 8: The learning curves of the PSO and the SAPSO with different parameter values.

Example 2-Prediction of the Chaotic Time Series

The Mackey-Glass chaotic time series $x(t)$ in consideration here is generated from the following delay differential equation:

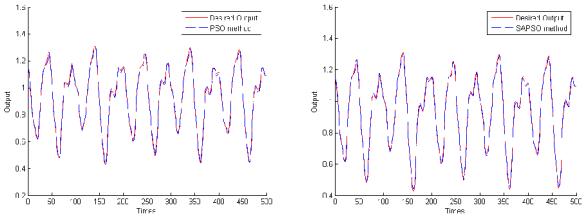
$$\frac{dx(t)}{dt}=\frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)}-0.1x(t) \qquad (29)$$

Crowder [34] extracted 1000 input-output data pairs $\{x, y^d\}$ which consisted of four past values of $x(t)$, i.e.

$$[x(t-18), x(t-12), x(t-6), x(t); x(t+6)] \qquad (30)$$

where τ=17 and $x(0)$=1.2. There are four inputs into the model, corresponding to the values of $x(t)$, and one output representing the value $x(t+\Delta t)$, where Δt is a time prediction into the future. The first 500 pairs (from $x(1)$ to $x(500)$) are the training data set, while the remaining 500 pairs (from $x(501)$ to $x(1000)$) are the testing data set used for validating the proposed method. The number of fuzzy rules is set to 6. The average best RMSE of the prediction output approximates 0.009 after 1000 generations.

Figures 9 (a) and (b) show the prediction results of PSO and SAPSO. Table 3 shows the comparison results of the prediction performance of all methods. Figure 10 shows the RMSE curves of the two models.



(a) (b)

Figure 9: The prediction results of (a) the PSO and (b) the SAPSO.

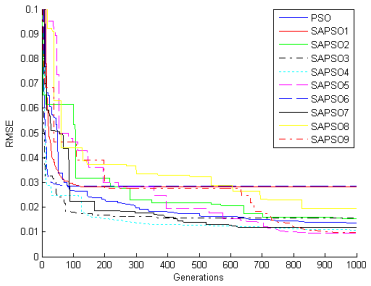


Figure 10: The learning curves of the PSO and the SAPSO with different parameter values for prediction problem.

Model \ RMSE	RMSE(Ave)	RMSE(Best)
Traditional PSO	0.012	0.006
SAPSO*1*	0.025	0.013
SAPSO*2*	0.015	0.010
SAPSO*3*	0.015	0.012
SAPSO*4*	**0.010**	**0.006**
SAPSO*5*	**0.010**	**0.007**
SAPSO*6*	0.029	0.012
SAPSO*7*	**0.011**	**0.008**
SAPSO*8*	0.019	0.010
SAPSO*9*	**0.009**	**0.006**

Table 3: The performance comparison with two methods

Example 3-Approximation of the Piecewise Function

The piecewise function was studied by Zhang [35] and Xu [36] and is defined as:

$$f(x)=\begin{cases} -2.186x-12.864 & -10\le x<-2 \\ 4.246x & -2\le x<0 \\ 10e^{-0.05x-0.5}\sin[(0.03x+0.7)x] & 0\le x\le 10 \end{cases} \quad (31)$$

over the domain D = [-10, 10]. The piecewise function is continuous and can be analyzed. However, traditional analytical tools are inadequate and often fail. This failure may be due to two reasons, namely, the wide-band information hidden at the turning points and the amalgamation of linearity and nonlinearity.

In this example, 200 training input patterns are uniformly generated from Eq. (31). Seven fuzzy rules are generated in this example. The RMSE curve is shown in Fig. 11 with all methods. Figures 12 (a)-(b) show the outputs of the function *f* with the PSO method and the SAPSO9 method. The solid line represents the output of function *f*, and the dotted line represents the approximation of various methods. The results comparing our model with PSO are tabulated in Table 4.

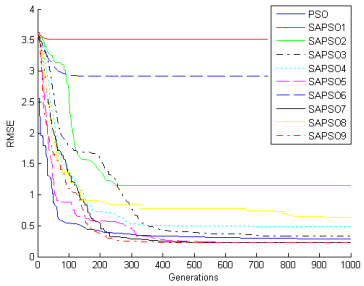
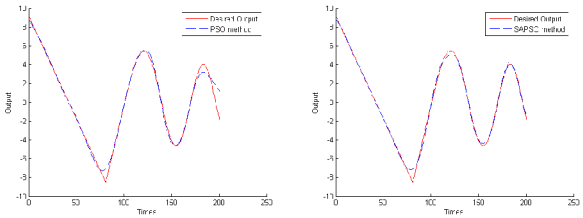


Figure 11: The learning curves of the PSO and the SAPSO with different parameter values for piecewise problem.



(a) (b)

Figure 12: The results of approximation using (a) the PSO method and (b) the SAPSO9 method.

Model \ RMSE	RMSE(Ave)	RMSE(Best)
Traditional PSO	0.28	0.12
SAPSO*1*	3.35	3.15
SAPSO*2*	1.15	0.45
SAPSO*3*	0.33	0.16
SAPSO*4*	0.43	0.14
SAPSO*5*	**0.24**	**0.13**
SAPSO*6*	2.96	2.18
SAPSO*7*	**0.21**	**0.09**
SAPSO*8*	0.64	0.36
SAPSO*9*	**0.20**	**0.09**

Table 4: The performance comparison with two methods.

The average computation time per generation for three examples with the PSO and SAPSO is tabulated in Table 5. We only update the value of sub-particles in each sub-swarm for SAPSO, and the total adjusted parameters of SAPSO are less than that of PSO. Therefore, the computation time required by SAPSO is less than that by PSO.

Example / Model	Identification of Nonlinear Dynamic System	Prediction of the Chaotic Time Series	Approximation of the Piecewise Function
PSO	1.21	7.5	3.15
SAPSO	0.35	1.70	0.65

Table 5: The average computation time of three examples for the PSO and the SAPSO (Unit: sec).

7 Discussion

From the above experimental results, we find that the parameters $\kappa_1, \kappa_2, \kappa_3$ and δ affect the performance of SAPSO. In order to test the relationship between the search trajectory and the parameter δ, we use the same value of $\kappa_1, \kappa_2, \kappa_3$ and ω in SAPSO6, SAPSO7, SAPSO8, and SAPSO9. We define the collection degree (*CD*) of a sub-swarm for each generation as follows:

$$CD = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \| particle(i) - particle(j) \| \qquad (32)$$

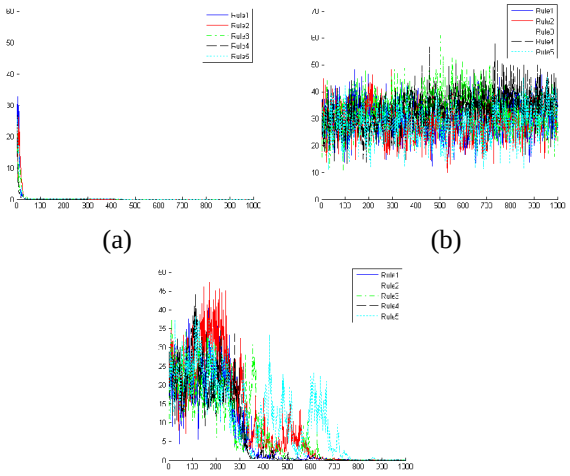
where N is the number of sub-swarms and $\| \ \|$ is the 2-norm (the Euclidean one for a vector). When *CD* is small, the particles are close to each other.

Figures 13 (a)-(c) and figures 14 (a)-(c) show the simulation results of example 1 when δ is 0.3, 0.7, and 0.6, diminishing to 0.4 gradually by increasing the learning steps. It is observed that the convergence speed of sub-particles in five sub-swarms is fastest when δ is 0.3, and the dynamics of five sub-swarms are always vibration when δ is 0.7, as seen in Fig. 13 (b).

The two situations mentioned are detrimental to the performance of the system. In Fig. 14 (a), the RMSE curves are sharp in the beginning and do not move afterwards. In Fig. 14 (b), the RMSE curves are like a ladder and drop slowly. Therefore, we hope to combine the advantages of the two situations mentioned and find the appropriate method for setting the parameters. We make the value of δ big enough to increase the chance of search in the beginning and then gradually reduce generations in number. We find that the performance of SAPSO5 and SAPSO9 is better than the others from the experiments. Moreover, from the above experimental results, the performance of the system is better when $\kappa_1 < \kappa_2 \le \kappa_3$.

8 Conclusion

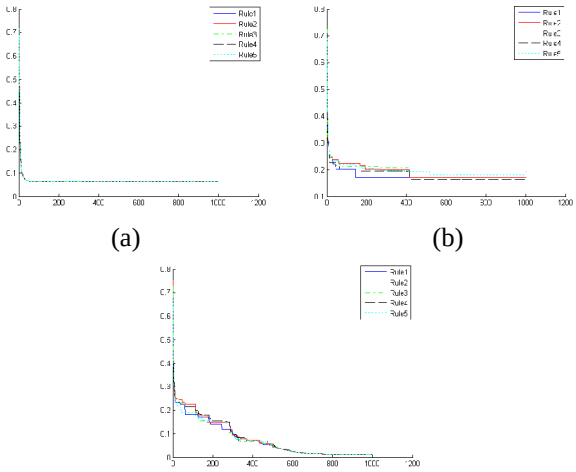
In this paper, a novel symbiotic adaptive particle swarm optimization (SAPSO) that adjusts the parameters of fuzzy systems was presented. The proposed SAPSO approach allows control over the dynamic characteristics of particles. The efficiency of the proposed SAPSO was demonstrated by its application to identification, prediction, and approximation of function problems. Simulation results show that the proposed SAPSO has better learning performance and less computation time requirements than the traditional PSO method.



(a) (b)

(c)

Figure 13: The collection degree for five sub swarm of example1 in learning processes when δ is (a) 0.3, (b) 0.7, and (c) 0.6~0.4.



(a) (b)

(c)

Figure 14: The RMSE curves for five sub-swarm of example1 in learning processes when δ is (a) 0.3, (b) 0.7, and (c) 0.6~0.4.

References

[1] C. T. Lin and C. S. G. Lee (1996), Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent System, NJ: Prentice-Hall.

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

[3] C. J. Lin and C. T. Lin (1997). An ART-based fuzzy adaptive learning control network. *IEEE Trans. on Fuzzy Systems*, vol. 5, no. 4, pp. 477-496.

[4] L. X. Wang and J. M. Mendel (1992). Generating fuzzy rules by learning from examples. *IEEE Trans. on Systems, Man, Cybern.*, vol. 22, no. 6, pp. 1414-1427.

[5] T. Takagi and M. Sugeno (1985). Fuzzy identification of systems and its applications to modeling and control. *IEEE Trans. on Systems, Man, Cybern.*, vol. SMC-15, pp. 116-132.

[6] J.-S. R. Jang (1993). ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Trans. on Systems, Man, and Cybern.*, vol. 23, pp. 665-685.

[7] C. F. Juang and C. T. Lin (1998). An on-line self-constructing neural fuzzy inference network and its applications. *IEEE Trans. on Fuzzy Systems*, vol. 6, no.1, pp. 12-31.

[8] F. J. Lin, C. H. Lin, and P. H. Shen (2001). Self-constructing fuzzy neural network speed controller for permanent-magnet synchronous motor drive. *IEEE Trans. on Fuzzy Systems*, vol. 9, no. 5, pp. 751-759.

[9] H. Takagi, N. Suzuki, T. Koda, and Y. Kojima (1992). Neural networks designed on approximate reasoning architecture and their application. *IEEE Trans. on Neural Networks*, vol. 3, no. 5, pp. 752-759.

[10] J. Kennedy and R. Eberhart (1995). Particle swarm optimization. *Proc. IEEE Int'l Conf. Neural Networks*, pp. 1942-1948.

[11] Z. L. Gaing (2004). A particle swarm optimization approach for optimum design of PID controller in AVR system. *IEEE Trans. on Energy Conversion*, vol. 19, Issue: 2, pp. 384-391.

[12] H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi (2000). A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE Trans. on Power Systems*, vol. 15, Issue: 4, pp. 1232-1239.

[13] M. A. Abido (2002). Optimal design of power-system stabilizers using particle swarm optimization. *IEEE Trans. on Energy Conversion,* vol. 17, Issue: 3, pp. 406-413.

[14] C. F. Juang (2004). A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Trans. on Systems, Man and Cybernetics, Part B*, vol. 34, Issue: 2, pp. 997-1006.

[15] R. Mendes, P. Cortez, M. Rocha, and J. Neves (2002). Particle swarms for feedforward neural network training. *Proc. Int'l Joint Conf. Neural Networks*, pp. 1895-1899.

[16] A. Savran (2007). an adaptive recurrent fuzzy system for nonlinear identification. *Applied Soft Comp.*, vol. 7, pp. 593-600.

[17] Y. Oysal and S. Yilmaz (2010). an adaptive wavelet network for function learning. *Neural Comp. Appli.*, vol. 19, pp. 383-392.

[18] B. Cetisli and A. Barkana (2010). Speeding up the scaled conjugate gradient algorithm and its application in neuro-fuzzy classifier training. *Soft Comt.*, vol. 14, no. 4, pp. 365-378.

[19] B. Cetisli (2010). Development of an adaptive neuro-fuzzy classifier using linguistic hedges: part 1. *Expert Syst. Appli.*, vol. 37, pp. 6093-6101.

[20] Y. Bodyanskiy, V. Kolodyazhniy, and A. Stephan (2001). An Adaptive Learning Algorithm for a Neuro-fuzzy Network. *Proc. Int'l Conf. 7th Fuzzy Days*, Dortmund, Germany, pp. 68-75.

[21] A. K. Palit and R. Babuska (2001). Efficient Training Algorithm for Takagi-Sugeno Type Neuro-Fuzzy Network. *Proc. IEEE Int'l Conf. Fuzzy Systems*, Vol. 3, pp. 1367-1371.

[22] R. Fletcher (reprinted 2006). *Practical methods of optimization*. Edn. 2nd, West Sussex: Wiley.

[23] P. E. Gill, W. Murray, and M. H. Wright (1981). *Practical Optimization*. London: Academic Press.

[24] R. Storn and K. V. Price (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. Global Opt.*, vol. 11, no. 4, pp. 341-359.

[25] A. E. Douglas (1994). *Symbiotic Interactions*. Oxford: Oxford University Press.

[26] C. J. Lin and C. F. Wu (2009). An Efficient Symbiotic Particle Swarm Optimization for Recurrent Functional Neural Fuzzy Network Design. *International Journal of Fuzzy Systems*, vol. 11, no. 4, pp. 262-271.

[27] C. H. Chen, C. J. Lin, and C. T. Lin (2009), Nonlinear System Control Using Adaptive Neural Fuzzy Networks Based on a Modified Differential Evolution. *IEEE Trans. on Systems, Man, and Cybernetics--Part C: Applications and Reviews,* vol. 39, no. 4, pp. 459-473.

[28] C. H. Chen, C. J. Lin, and C. T. Lin (2009). Using an Efficient Immune Symbiotic Evolution Learning for Compensatory Neuro-Fuzzy Controller. *IEEE Trans. on Fuzzy Systems*, vol. 17, no. 3, pp. 668-682.

[29] P. N. Suganthan (1999). Particle swarm optimizer with neighborhood operator. *Proc. Congress on Evolutionary Computation*, vol. 3.

[30] K. Yasuda, A. Ide, and N. Iwasaki (2003). Adaptive particle swarm optimization. *Proc. IEEE Int'l Conf. Systems, Man and Cybernetics*, vol. 2, pp. 1554-1559.

[31] D. E. Moriarty and R. Miikkulainen (1996). Efficient reinforcement learning through symbiotic evolution. *Mach. Learn.*, vol. 22, pp. 11-32.

[32] M. Clerc and J. Kennedy (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. on Evolutionary Computation*, vol. 6, Issue: 1, pp. 58-73.

[33] K. S. Narendra and K. Parthasarathy (1990), Identification and control of dynamical systems using neural networks. *IEEE Trans. on Neural Networks*, vol. 1, no. 1, pp. 4-27.

[34] R. S. Crowder (1990). Predicting the Mackey-Glass timeseries with cascade-correlation learning. *Proc. of the 1990 Connectionist Models Summer School*, Carnegie Mellon University, pp. 117–123.

[35] Q. Zhang and A. Benveniste (1992). Wavelet Networks. *IEEE Trans. on Neural Networks,* pp. 889-898.

[36] D.W.C. Ho, P. A. Zhang, and J. Xu (2001). Fuzzy Wavelet Networks for Function Learning. *IEEE Trans. on Fuzzy Systems*, vol.9, pp. 200-211.