Expert Systems with Applications 36 (2009) 5362–5370

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A self-generating fuzzy system with ant and particle swarm cooperative optimization

Chia-Feng Juang*, Chi-Yen Wang

Department of Electrical Engineering, National Chung Hsing University, Taichung 402, Taiwan, ROC

ARTICLE INFO

Keywords:
Swarm intelligence
Ant colony optimization
Particle swarm optimization
Fuzzy clustering
Fuzzy control

ABSTRACT

This paper proposes a self-generating fuzzy system with a learning ability from a combination of the on-line self-aligning clustering (OSAC) algorithm and ant and particle swarm cooperative optimization (APSCO). The proposed OSAC algorithm not only helps generate rules from on-line training data, but also helps avoid generating highly overlapping fuzzy sets. Once a new rule is generated, APSCO optimizes the corresponding antecedent and consequent parameters. In APSCO, ant colony and particle swarm coexist in a population, and both search for an optimal parameter solution simultaneously in each iteration. Ant paths not only help determine the consequent parameters of generated rules, they also help generate auxiliary particles. Well-performing particles are selected from the auxiliary particles and original particles. And these selected particles cooperate to find a better solution through particle swarm optimization. This paper applies the proposed self-generating fuzzy system to different fuzzy controller design problems, and compares it with other genetic and swarm intelligence algorithms and their hybrids to verify system performance.

1. Introduction

The advent of bio-inspired computation techniques has inspired new methods for solving optimization problems, including the optimal design of fuzzy systems and neural networks. Evolutionary computation (EC) (Yao, 1999) and swarm intelligence (SI) (Kennedy, Eberhart, & Shi, 2001) are two well-known bio-inspired optimization techniques. The EC technique is heuristic and stochastic. It involves populations of individuals with evolutionary operators similar to a biological phenomenon, like crossover and mutation. Genetic algorithms (GAs) (Michalewicz, 1999) are the most well-known EC techniques. The SI technique studies collective behavior in decentralized systems. Its development was based on mimicking the social behavior of animals or insects in an effort to find the optima in the problem space. In SI, a population of simple agents interacts locally with one other and with their environment. In contrast to EC, SI models do not explicitly use evolutionary operators.

Particle swarm optimization (PSO) (Clerc & Kennedy, 2002; Kennedy & Eberhart, 1995) and ant colony optimization (ACO) (Dorigo & Gambardella, 1997; Dorigo & Stützle, 2004; Dorigo, Maniezzo, & Colorni, 1996) are two well-known SI algorithms. The PSO algorithm was developed from observations of the social behavior of animals, including bird flocking and fish schooling. An individual in PSO is assigned with a randomized velocity according to its own and its companions' experiences. The individuals, called particles, then fly through hyperspace to find good solutions. The ACO technique is inspired by real ant colony observations. It is a multi-agent approach to solving difficult combinatorial optimization problems such as the traveling salesman problem (TSP) (Dorigo & Gambardella, 1997; Dorigo et al., 1996). In the ACO meta-heuristic, artificial ant colonies cooperate in finding good solutions for difficult discrete optimization problems. Successful applications of PSO and ACO to several optimization problems have demonstrated their potential (Heo, Lee, & Ramirez, 2006; Ho, Yang, Ni, & Wong, 2006; Mendes, Cortez, Rocha, & Neves, 2002; Parpinelli, Lopes, & Freitas, 2002; Sim & Sun, 2003).

The design of fuzzy systems can be regarded as an optimization problem. To ease design efforts and improve system performance, researchers have proposed designs using EC and SI techniques (Belarbi & Titel, 2000; Cassillas, Cordon, Viana, & Herrera, 2000; Chou, 2006; Chung, Lin, & Lin, 2000; Giordano, Naso, & Turchiano, 2006; Homaifar & McCormick, 1995; Juang, 2002, 2004; Juang, Lin, & Lin, 2000; Wong & Her, 1999). In contrast to neural fuzzy systems, these techniques can be applied to design problems where precise input–output training data pairs are unavailable or costly to obtain, such as fuzzy control problems. Fuzzy system design consists of structure and parameter designs, where structure learning determines the number of fuzzy rules and fuzzy sets in each input variable and parameter learning determines the free parameter in the antecedent and consequent parts. Most studies

* Corresponding author.
E-mail address: cfjuang@dragon.nchu.edu.tw (C.-F. Juang).

0957-4174/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2008.06.101

that used genetic algorithms (GAs) or swarm intelligence (SI) for fuzzy system optimization used either pre-defined and fixed structures (Belarbi & Titel, 2000; Chung et al., 2000; Homaifar & McCormick, 1995; Juang, 2002, 2004; Juang et al., 2000) or defined the number of membership functions for each input variable in advance and searched the significant rules from a huge rule-base (Cassillas et al., 2000; Chou, 2006; Giordano et al., 2006; Wong & Her, 1999). In contrast to those studies, this paper presents a self-generating fuzzy system whose structure is designed through a newly proposed on-line clustering method. Though many types of fuzzy clustering have been proposed (Höppner, Klawonn, Kruse, & Runkler, 1999), they are primarily used in neural fuzzy systems (Lin & Lee, 1996). This paper incorporates a new fuzzy clustering method, the on-line self-aligning clustering (OSAC) method, into the proposed cooperative SI model. Most clustering methods are performed either with training data collected in advance or cluster numbers assigned *a priori* (Höppner et al., 1999). These clustering methods cannot be used for design problems where data is generated only when on-line operations are performed, like the control problems simulated in this paper. For this reason, the proposed OSAC algorithm is combined with an on-line automatic rule generation ability. In addition, the self-aligning operation of the clusters in the OSAC helps avoid generating highly overlapping fuzzy sets and reduces the total number of fuzzy sets.

Previous studies have proposed parameter learning of fuzzy systems using PSO (Juang, 2004) or ACO (Cassillas et al., 2000). The initial positions of each particle affect the performance of PSO, and bad initialization may cause poor performance. One study (Cassillas et al., 2000) uses ACO in fuzzy system design where learning data is collected off-line in advance. The consequent of each fuzzy rule is selected only from a discrete space by ACO in Cassillas et al. (2000) and the antecedent part parameters are not tuned. For problems where high accuracy is a major concern, the fixed antecedent parameters and transition from continuous search space to discrete search space degrade fuzzy system performance. Simulations in Section 5 verify this fact. This paper optimizes parameters of each newly generated rule by OSAC in the proposed self-generating fuzzy system using a cooperative SI model, the ant and particle swarm cooperative optimization (APSCO). In APSOC, ACO first finds the consequent parameters in the discrete-space domain of the rules whose antecedent parts are determined from OSAC. Searching in the discrete-space domain by ants helps to find good solutions. However, the search constraint in a discrete-space domain restricts learning accuracy, and the ants do not optimize antecedent part parameters. To overcome this problem, APSCO also takes the advantage of PSO searching abilities. PSO simultaneously searches both the rule antecedent and consequent part parameters in the continuous domain. This PSO search ability compensates for the aforementioned weakness of ACO in fuzzy system design problems. Simulations show that the cooperation of PSO and ACO in the proposed APSCO approach can produce good performance.

This paper is organized as follows. Section 2 describes the fuzzy systems to be designed by APSCO and the basic concepts of ACO and PSO. Section 3 introduces the OSAC algorithm in the self-generating fuzzy system. Section 4 introduces APSCO for parameter learning. Section 5 simulates three examples and compares the performance of the proposed design method with other design methods. Finally, Section 6 draws conclusions.

2. Fuzzy system and swarm intelligence

Section 2.1 describes the fuzzy system to be designed in this study. Since the proposed ant and particle swarm cooperative optimization (APSCO) used the ideas of ant colony optimization (ACO) and particle swarm optimization (PSO) techniques in swarm intelligence, basic concepts of ACO and PSO are described in Sections 2.2 and 2.3, respectively.

2.1. Fuzzy systems

The ith rule, denoted as R_i, in the fuzzy system is represented in the following form:

$$R_i: \text{If } x_1(k) \text{ is } A_{i1} \quad \text{And} \quad \ldots \quad \text{And} \quad x_n(k) \text{ is } A_{\text{in}} \quad \text{Then } y(k) \text{ is } a_i \tag{1}$$

where k is time step, $x_1(k),\ldots,x_n(k)$ are input variables, $y(k)$ is the system output variable, A_{ij} is a fuzzy set, and a_i is a crisp value. Fuzzy set A_{ij} uses a Gaussian membership function

$$M_{ij}(x_j) = \exp\left\{-\left(\frac{x_j - m_{ij}}{\sigma_{ij}}\right)^2\right\} \tag{2}$$

where m_{ij} and σ_{ij} represent the center and width of the fuzzy set A_{ij}, respectively. The fuzzy AND operation in the inference engine is implemented by the algebraic product in fuzzy theory. Thus, given an input data set $\vec{x} = (x_1,\ldots,x_n)$, the firing strength $\phi_i(\vec{x})$ of rule i is calculated by

$$\phi_i(\vec{x}) = \prod_{j=1}^{n} M_{ij}(x_j) = \exp\left\{-\sum_{j=1}^{n}\left(\frac{x_j - m_{ij}}{\sigma_{ij}}\right)^2\right\} \tag{3}$$

If there are r rules in a fuzzy system, the output of the system calculated by the weighted average defuzzification method is

$$y = \frac{\sum_{i=1}^{r}\phi_i(\vec{x})a_i}{\sum_{i=r}^{r}\phi_i(\vec{x})} \tag{4}$$

2.2. Ant colony optimization (ACO)

In ACO Dorigo and Stützle (2004), a finite size colony of artificial ants is created. Each ant builds a solution. While building its own solution, each ant collects information based on the problem characteristics and its own performance. The performance measure is based on a quality function $F(\cdot)$. The ACO method can be applied to discrete combinational problems, where solutions to the optimization problem can be expressed in terms of feasible paths on a graph. Among all feasible paths, ACO aims to locate the one with a minimum cost. The problem of selecting the consequent of fuzzy rules can be designed as a combinational problem and solved by ACO. The information collected by the ants during the search process is stored in pheromone trails τ associated to the connection of all edges. The ants cooperate in finding the solution by exchanging information via the pheromone trials. Edges can also have an associated heuristic value η. The η value represents *a priori* information about the problem instance definition or run-time information provided by a source different from the ants. The heuristic information is auxiliary in ACO and ACO works even without the use of it. Once all ants have completed their tours (i.e., at the end of the each iteration), ACO algorithms update the pheromone trails using all the solutions produced by the ant colony.

The whole ACO algorithm can be described by taking the traveling salesman problem (TSP) as an example. The TSP is to find a minimal length with each city visited once. We are given a set of N cities, represented by nodes, and a set E of edges with fully connecting nodes N. Let d_{ij} be the length of the edge $(i,j) \in E$, that is the distance between cities i and j, with $i,j \in N$. At each iteration t, an ant in city i has to choose the next city j to head for from among those cities that it has not yet visited. The probability of picking a certain city j is calculated using the distance between cities i

and j, and the amount of pheromone on the edge between these two cities. The first algorithm to fall within the ACO algorithms framework was the ant system (AS) Dorigo et al. (1996). In AS algorithm, the probability with which an ant q chooses to go from city i to city j is

$$p_{ij}^{q}(t)=\begin{cases}\frac{[\tau_{ij}(t)][\eta_{ij}]^{\beta}}{\Sigma_{\ell\in N_i^q}[\tau_{i\ell}(t)][\eta_{i\ell}]^{\beta}} & \text{if } j\in N_i^q\\ 0 & \text{otherwise}\end{cases} \tag{5}$$

where $\tau_{ij}(t)$ is the amount of pheromone trails on edge (i,j) at iteration t, $\eta_{ij}=1/d_{ij}$ is the heuristic value of moving from city i to city j, N_i^q is the set of neighbors of city i for the qth ant, and parameter β controls the relative weight of pheromone trail and heuristic value. After all ants have completed their tours, the pheromone level is updated by

$$\tau_{ij}(t+1)=(1-\rho)\tau_{ij}(t)+\Delta\tau_{ij}(t) \tag{6}$$

where $0<\rho\leqslant 1$ is the pheromone trail evaporation rate. The term $\Delta\tau_{ij}(t)$ is related to the performance of each ant. Details of this term when using the AS can be found in Dorigo et al. (1996). Since the AS proposal, many other ACO algorithms, like ant colony system (Dorigo & Gambardella, 1997) and MAX MIN ant system (Stützle & Hoos, 2000), have been studied (Dorigo & Stützle, 2004). The major differences between these ACO algorithms and AS are the probability selection techniques or pheromone update. The pseudocode for a basic ACO algorithm is as follows.

Algorithm ACO
Initialize parameters and pheromone trails
While (stop conditions not satisfied) do
{For $i=1$ to P_S do
{Generate ant paths for solution construction using Eq. (5)
Find quality function $F(\cdot)$
}
Update pheromone trails using Eq. (6)
}

2.3. Particle swarm optimization (PSO)

The swarm in PSO is initialized with a population of random solutions (Clerc & Kennedy, 2002; Kennedy & Eberhart, 1995). Each potential solution is called a particle. Each particle has a position represented by a position vector $\vec{s}_i$. A swarm of particles moves through the problem space, with the velocity of each particle represented by a velocity vector $\vec{v}_i$. At each time step, a function f is evaluated using $\vec{s}_i$ as an input. Each particle keeps track of its own best position, which is associated with the best fitness it has achieved so far in a vector $\vec{p}_i$. Furthermore, each particle is defined within the context of a topological neighborhood comprising itself and some other particles in the population. The best position found by any member of the neighborhood is tracked as $\vec{p}_{ig}$. For a typical PSO (Shi & Eberhart, 1998), a new velocity for particle i is updated as follows:

$$\vec{v}_i(t)=w\vec{v}_i(t-1)+c_1\phi_1(\vec{p}_i(t-1)-\vec{s}_i(t-1))+c_2\phi_2(\vec{p}_{ig}(t-1)-\vec{s}_i(t-1)) \tag{7}$$

where w is inertia weight, c_1 and c_2 are positive constants, ϕ_1 and ϕ_2 are uniformly distributed random numbers in $[0,1]$. The term $\vec{v}_i$ is limited to the range $[-\vec{v}_{\max},\vec{v}_{\max}]$. Depending on their velocities, each particle changes its position according to the following formula:

$$\vec{s}_i(t)=\vec{s}_i(t-1)+\vec{v}_i(t), \tag{8}$$

The particle population tends to cluster together around the best solution. The pseudocode of a basic PSO algorithm is as follows:

Algorithm PSO
Initialize parameters and randomly generate an initial population
While (stop conditions not satisfied) do
{For $i=1$ to population size do
{Find $f(\vec{s}_i)$
If $f(\vec{s}_i)<f(\vec{p}_i)$ set $\vec{p}_i=\vec{s}_i$
}
Find $\vec{p}_{ig}$
Velocity update using Eq. (7)
Position update using Eq. (8)
}
}

3. Structure learning by on-line self-aligning clustering

The proposed self-generating fuzzy system uses an on-line self-aligning clustering (OSAC) algorithm to determine the numbers of rules and fuzzy sets in each input variable. The OSAC algorithm also decides the initial positions of fuzzy sets. As stated in Juang and Lin (1998), a rule geometrically corresponds to a fuzzy cluster in the input space. For each incoming input pattern $\vec{x}$, the firing strength can be regarded as the degree to which the incoming input pattern belongs to the corresponding cluster. According to this concept, the firing strength $\phi_i(\vec{x})$ in Eq. (3) serves as the criterion for deciding if a new fuzzy rule should be generated. For each incoming data $\vec{x}(k)$, find

$$I=\arg\max_{1\leqslant i\leqslant r(k)}\phi_i(\vec{x}(k)) \tag{9}$$

where $r(k)$ is the number of existing rules at time k. If $\phi_I\leqslant\phi_{th}$ or $r(k)=0$, where $\phi_{th}\in(0,1)$ is a pre-specified threshold, a new fuzzy rule is generated and $r(k+1)=r(k)+1$. This rule generation criterion is the same as that proposed in Juang and Lin (1998). However, the generation of fuzzy sets in each input variable and their initial shapes assignment methods in the OSAC in this paper are different from Juang and Lin (1998).

When the OSAC generates a new fuzzy rule, the next step is to decide whether or not to generate a new fuzzy set on each input variable. One simple method is to assign a new fuzzy set in each input variable for each newly generated rule. However, this approach may generate highly-overlapping fuzzy sets, as Fig. 1a indicates. Fig. 1b shows that merging these highly-overlapping fuzzy sets generates aligned fuzzy clusters, which helps reduce the number of fuzzy sets and increase system interpretability. In Juang and Lin (1998), a fuzzy set similarity measure method is used to generate aligned fuzzy clusters. However, the similarity measure opera-

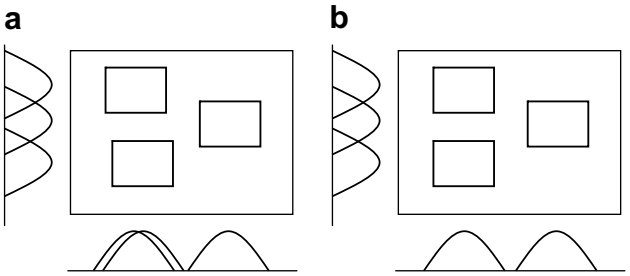

Fig. 1. (a) Flexible partition, where some fuzzy sets are highly overlapping. (b) Flexible partition by fuzzy clustering with the elimination of highly overlapping fuzzy sets.

tion is very complex. This paper proposes a simple but effective criterion. In the OSAC, the one-dimensional matching degree $M_{ij}(x_j)$ of each input directly serves as the criterion for deciding if a new fuzzy set should be generated. For each input dimension, compute

$$I_j = \arg \max_{1 \leqslant i \leqslant h_j(k)} M_{ij}(x_j(k)), \quad j = 1, \ldots, n \tag{10}$$

where $h_j(k)$ is the number of fuzzy sets in the jth input variable. If $M_{I_j j}(x_j) < s_{th}$ or $h_j(k) = 0$, a new fuzzy set is generated in the j th input variable and $h_j(k+1) = h_j(k) + 1$. The value s_{th} is a pre-specified threshold and $0 < s_{th} < 1$. The center and width of the new fuzzy set are shown below

$$m_{h_j(k+1)j} = x_j, \sigma_{h_j(k+1)j} = \sigma_{\text{fixed}} \tag{11}$$

where σ_{fixed} is a pre-defined threshold and is set at 0.4 in this paper. Otherwise, the fuzzy set $A_{I_j j}$ serves as the antecedent part of the new rule on input variable j. That is, a self-alignment operation is performed. The measure criterion using Eq. (6) helps avoid generating highly overlapping fuzzy sets and reduces computation costs. The OSAC algorithm is summarized as follows:

On-line self-aligning clustering (OSAC) algorithm
For each newly incoming data $\underline{x}$ do
{Compute Eq. (5)
IF ($\phi_I < \phi_{th}$ or $r(k) = 0$) THEN
{$r(k+1) = r(k) + 1$
For $j = 1 \sim n$
{Compute Eq. (6)
IF ($M_{I_j j}(x_j) < s_{th}$ or $h_j(k) = 0$)
THEN
{generate a new fuzzy set, $h_j(k+1) = h_j(k) + 1$,
set center and width of the new fuzzy set by Eq. (11)
}
ELSE
{use the fuzzy set $A_{I_j j}$ as the antecedent part of rule $r(k+1)$}
}
}}

4. Parameter learning by APSCO

4.1. Learning flow of APSCO

Ant and particle swarm cooperative optimization (APSCO) tunes the parameters of each newly generated fuzzy rule. The APSCO algorithm searches through a constant population of individuals. Each individual represents a parameter solution of the fuzzy system described in Section 2.1. That is, each individual has the following form:

$$[m_{11}, \sigma_{11}, \ldots, m_{1n}, \sigma_{1n}, u_1, \ldots, m_{r1}, \sigma_{r1}, \ldots, m_{rn}, \sigma_{rn}, u_r] \tag{12}$$

where an individual contains a total of $r(2n+1)$ terms. An individual may be generated by an ant path or a particle. Individuals generated by ants and particles are called ant individuals and particle individuals, respectively. Fig. 2 shows a block diagram of the APSCO algorithm. Generation of population individuals are introduced as follows.

4.1.1. Population initialization

Suppose a population contains $2N$ individuals. Initially (the first iteration), on-line self-aligning clustering (OSAC) and N different ant paths generate half of the population, where OSAC and an ant path determine the rule antecedent and consequent part parameters, respectively. The N ant individuals contain no rules initially. Rules are generated on-line as the OSAC is performed on the on-line generated data during the evaluation of these N ant individuals. If a new cluster is generated during the evaluation of the ith ant individual, then the rule number in this and other unevaluated ant individuals (individuals $i+1, \ldots, N$) also increases by one, with the consequent value being selected by ant paths. The other N individuals in the first iteration are generated as particles. If the number of clusters after evaluation of the N ant individuals is r_1, then the number of rules in the N particle individuals are all equal to r_1. For initial particle position generation, the antecedent parameters are generated by adding small random values to these r_1 clusters. The consequent parameters are randomly generated from the output search range. The performance of these particles is then evaluated. Since the objective of using PSO is to optimize both the antecedent and consequent parameters in existing fuzzy rules, the OSAC is not conducted during the particle performance evaluation.

The second and subsequent iterations generate a new population with $2N$ new individuals. The following section introduces the generation of ant and particle individuals in these iterations:

- *Ant individuals generation*. The antecedent part parameters of all ant individuals are determined by OSAC. The consequent parameters of existing and new clusters (rules) are determined by ant paths. This way, N ant paths generate N ant individuals (fuzzy systems). During the new ant individual generation process, some ants may choose the same path and generate the same

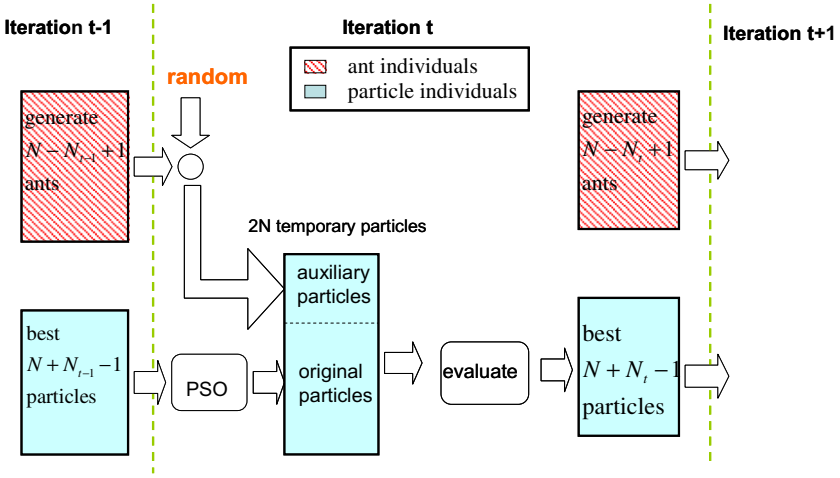

Fig. 2. Block diagram of the proposed ant and particle swarm cooperative optimization (APSCO).

individuals. This phenomenon becomes more obvious as more iterations are conducted due to pheromone matrix convergence. To consider this phenomenon, suppose that N_t ants have the same path at the tth iteration. Then only $N - N_t + 1$ different ant individuals are reserved in the population. The performance of these $N - N_t + 1$ individuals is evaluated and the pheromone matrix is updated according to their performance.

- *Particle individuals generation.* The remaining $N + N_t - 1$ new individuals in the population (population size is fixed at $2N$) in iteration t are partly generated by particles in the last iteration. The original $N + N_{t-1} - 1$ particle individuals from the last iteration are optimized by PSO. The performance of these optimized particle individuals is then evaluated. In addition to these optimized particle individual, the $N - N_{t-1} + 1$ ant individuals in the previous iteration (iteration $t - 1$) also help generate auxiliary particles to improve particle search performance. Adding these ant individuals with small random values generates auxiliary particles. The purpose of adding small random values is to distinguish auxiliary particles from existing ant individuals and to improve the algorithm's exploration ability. Suppose an original individual has the form in Eq. (12), then the auxiliary particle takes the following form

$$[m_{11} + \Delta m_{11}, \sigma_{11} + \Delta\sigma_{11}, \ldots, m_{1n} + \Delta m_{1n}, \sigma_{1n} + \Delta\sigma_{1n}, u_1 + \Delta u_1, \ldots, m_{rn} + \Delta m_{rn}, \sigma_{rn} + \Delta\sigma_{rn}, u_r + \Delta u_r] \quad (13)$$

If the input domain of x_i is $[Lx_i, Ux_i]$, then Δm_{ij} and $\Delta\sigma_{ij}$ are randomly and uniformly generated from the interval $\frac{1}{10} \cdot [Lx_i, Ux_i]$. The same method generates Δu_i according to the output variable range. That is, if the output range of y is $[Lu, Uu]$, than Δu_i is randomly and uniformly generated from the interval $\frac{1}{10} \cdot [Lu, Uu]$. The performance of these $N - N_{t-1} + 1$ auxiliary particles is then evaluated. These auxiliary particles together with the original $N + N_{t-1} - 1$ particles constitute a total of $2N$ temporary particles. Only the best $N + N_t - 1$ particles are reserved from among these $2N$ particles. In the next iteration, these reserved particles cooperate to find better solutions through PSO.

Due to OSAC structure learning, the rule number r_t in iteration t may be different from the rule number r_{t-1} in iteration $t - 1$. This phenomenon usually happens in the first few iterations. For this case, the following operations are used to insure all particles, including original and auxiliary particles, to be of the same length in iteration t. Insert $r_t - r_{t-1}$ rules into the original particles in iteration $t - 1$ before performance evaluation. Here, the antecedent parameters are generated by adding small random values to clusters $r_{t-1} + 1, \ldots, r_t$ and the consequent parameters are randomly generated from the output variable range.

The rule number in some of the $N - N_{t-1} + 1$ ant individuals may be smaller than r_t. For this case, new rules are inserted into ant individuals whose rule number is smaller than r_t so that all generated auxiliary particles also have r_t rules. The parameters of these newly inserted rules are determined as in the generation of ant individuals described previously. That is, existing clusters and new ant paths determine the antecedent and consequent parameters, respectively.

Fig. 3 shows a flowchart of the whole learning process. The learning process ends when a pre-defined number of iterations is performed. The following two subsections introduce details of individuals updated by ACO and PSO.

4.1.2. Individual update by ACO

For the individual generated by clustering and an ant path, the antecedent parameters m_{ij} and σ_{ij} in Eq. (12) are determined by Eq. (11), and the consequent parameters u_i are determined by an ant path. In this subsection, application of ACO to the design of the consequent value u_i of each generated rule (cluster) is introduced. First, all of the possible consequent actions are listed in a set $U = \{a_1, \ldots, a_M\}$. That is, the number of candidate consequent values of u_i in Eq. (1) is M for each rule and $u_i \in U$. Suppose there are r rules, then the complexity of finding the best consequent combination is M^r. This arrangement transfers the consequent parameter design problem into a discrete combinational optimization problem so that an ACO algorithm can be exploited to find the solution. The path of an ant is regarded as one combination of consequent values selected from every rule. For example, Fig. 4 shows three fuzzy rules (denoted by R_1, R_2, and R_3) in a fuzzy system and four candidate consequent values, $a_1, \ldots, a_4$, for each rule. These rules are generated by OSAC. Starting from the initial state, an ant moves through R_1 and R_2, and stops at R_3. A bold line marks the ant path. For each rule, the node visited by the ant is selected as the consequent value of the rule. For the whole fuzzy system constructed by the ant in Fig. 4, the consequent values in R_1, R_2, and R_3 are a_2, a_4, and a_1, respectively. Selection of the consequent value is based on pheromone trails between each rule. The size of the pheromone matrix is $r \times M$ and each entry in the matrix is denoted by τ_{ij}, where $i = 1, \ldots, r$ and $j = 1, \ldots, M$. As shown in Fig. 4, when the ant arrives at rule R_m, selection of the M candidate actions (denoted by nodes) of R_{m+1} is according to τ_{m+1j}, $j = 1, \ldots, M$.

The above analysis reveals that the path of an ant corresponds to the choice of the consequent values of all rules, according to the pheromone level. At iteration t, the transition probability for selecting the consequent value a_j in the fuzzy rule i is defined by

$$p_{ij}(k) = \frac{\tau_{ij}(t)}{\sum_{z=1}^{M} \tau_{iz}(t)} \quad (14)$$

where $i = 1, \ldots, r$ and $j = 1, \ldots, M$. This study assumes that no *a priori* information about the problem is known for computing heuristic information η. For example, in an unknown plant control problem, it is assumed that no *a priori* knowledge about the controlled plant is known and no training data is available before learning.

When an ant completes a path, the corresponding fuzzy system (individual) is evaluated and a quality value F is computed. For the simulated control examples in this paper, F is defined as the inverse of control error. The pheromone trails, τ_{ij}, are updated according to F. For each iteration, after all the ants in the colony have completed their paths, i.e., after the construction of N fuzzy systems, select the one with the highest F from the initial iteration until now. If a new global best ant is found in this iteration, then pheromone trails on the path traveled by the global best ant are updated; otherwise, no pheromone update is performed in this iteration. Denote the global best ant as q^* with the corresponding quality value as F_{q^*}. The new pheromone trail $\tau_{ij}(t+1)$ is updated by

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t), \text{if } (i,j) \in \text{global-best-path} \quad (15)$$

where $0 < \rho < 1$ is the pheromone trail evaporation rate ($\rho = 0.1$ in this paper) and

$$\Delta\tau_{ij}(t) = F_{q^*} \quad (16)$$

4.1.3. Individual update by PSO

In iteration t, PSO updates all of the $N + N_{t-1} - 1$ particles generated either from auxiliary particles or original particle individuals in the previous iteration. The position and velocity of a particle i is denoted by $\vec{s}_i$ and $\vec{v}_i$, respectively. According to particle performance, it is possible to find individual best position $\vec{p}_i$ of each particle and its neighborhood best particle $\vec{p}_{ig}$. If a particle i is generated from auxiliary particles or by inserting new rule(s) into an existing particle, this particle is regarded as a newly generated particle and its individual best $\vec{p}_i$ is equal to its initial position $\vec{s}_i$.

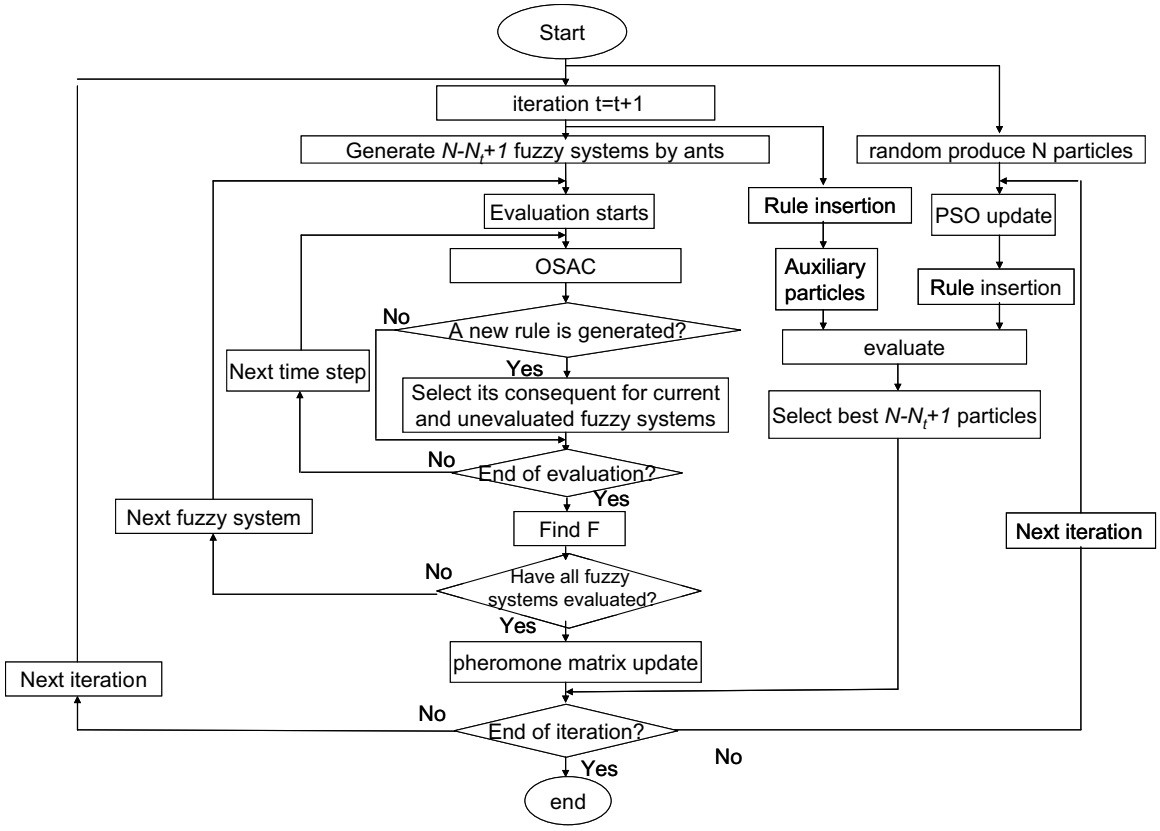

Fig. 3. Flow chart of the proposed self-generating fuzzy system.

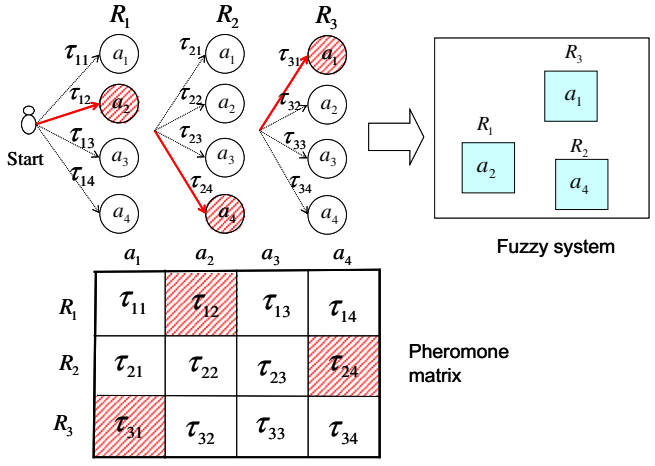

Fig. 4. The relationships between pheromone matrix, ant paths, and selected rules consequent.

All of the $N + N_{t-1} - 1$ particles comprise r_{t-1} rules. The neighbors of a particle for finding $\vec{p}_{ig}$ are defined as the particles that also have r_{t-1} rules, which can be regarded as a local version of PSO. The velocity $\vec{v}_i$ of each particle i is updated using its individual best position, $\vec{p}_i$, and the neighborhood best position, $\vec{p}_{ig}$. Here, the following equation with a constriction coefficient (Clerc & Kennedy, 2002) is used for velocity updating

$$\vec{v}_i(t) = \chi(\vec{v}_i(t-1) + c_1\phi_1(\vec{p}_i(t-1) - \vec{s}_i(t-1)) + c_2\phi_2(\vec{p}_{ig}(t-1) - \vec{s}_i(t-1))), \quad i = 1,\ldots,N+N_{t-1}+1 \tag{17}$$

where c_1 and c_2 are positive constants, ϕ_1 and ϕ_2 are uniformly distributed random numbers in $[0,1]$, and χ controls the magnitude of $\vec{v}$. With parameter χ, it is not necessary to set an upper limit on $\vec{v}$. This paper sets parameters as $c_1 = 1$, $c_2 = 1$, and $\chi = 0.8$ as in Juang (2004). These parameters were chosen because they obtained the best performance over several trials. Depending on their velocities, each particle changes its position according to the following formula:

$$\vec{s}_i(t) = \vec{s}_i(t-1) + \vec{v}_i(t), \quad i = 1,\ldots,N+N_{t-1}+1 \tag{18}$$

5. Simulations

This section simulates fuzzy system design using OSAC and APSCO. In APSCO, the population size is set at $2N = 40$ for different examples. Simulation examples include nonlinear plant tracking control and reversing a truck following a circular path. These examples generate training data only when fuzzy control starts. Only the desired control trajectory is available, and the desired fuzzy controller outputs are unknown in advance, so fuzzy neural networks with supervised learning cannot be directly applied to these problems. These examples compare the performance of the proposed design method with other evolutionary and swarm intelligence methods, and their hybrids.

5.1. Example 1

In this example, the plant to be controlled is described by

$$y_p(k+1) = \frac{y_p(k)y_p(k-1)(y_p(k)+2.5)}{1+y_p^2(k)+y_p^2(k-1)} + u(k), \quad -1.2 \leqslant y_p \leqslant 1.2 \tag{19}$$

where $u(k)$ is the control input and $-1.2 \leqslant u(k) \leqslant 1.2$. The objective is to control the output $y_p(k)$ to track the following desired trajectory:

$$y_d(k+1) = 0.2 \cdot (0.6y_d(k) + 0.2y_d(k-1) + r(k)), \quad 0 \leqslant k < 250 \tag{20}$$

where

$$r(k) = 0.2\sin(2\pi k/25) + 0.4\sin(\pi k/32) \tag{21}$$

Fuzzy controller inputs are $y_d(k)$, $y_p(k-1)$, and $y_p(k)$. The control error is defined as

$$\text{SAE} = \sum_{k=1}^{250} |e(k)| \tag{22}$$

In OSAC, parameter ϕ_{th} for fuzzy clustering is set at 0.05. Two values of s_{th} ($s_{\text{th}} = 0$ and 0.8) are simulated. In APSCO, the set of candidate control actions is $U = \{0, \pm 0.1, \pm 0.2, \cdots, \pm 1.2\}$, where the output range is uniformly divided into 25 candidate actions. The pheromone update quality value is $F = 100/\text{SAE}$. The number of iterations is 3000, with 50 runs. The average number of fuzzy rules generated over these 50 runs is 4. The numbers of fuzzy sets with $s_{\text{th}} = 0$ and $s_{\text{th}} = 0.8$ are 12 and 11, respectively. An average of 3, 4, and 4 fuzzy sets occur in inputs $y_d(k)$, $y_p(k-1)$, and $y_p(k)$, respectively, when $s_{\text{th}} = 0.8$. Fig. 5 illustrates the final distribution of fuzzy sets in the three inputs after learning from one of the 50 runs with $s_{\text{th}} = 0.8$. Table 1 shows the learning average and STD of errors with and without the self-aligning operation. The self-generating operation helps reduce the total number of fuzzy sets with only slight training error increase. Fig. 6 shows one control result.

This example compares the elite GA (EGA) (Juang, 2002), constriction PSO (CPSO) (Clerc & Kennedy, 2002), a hybrid of GA and PSO (HGAPSO) (Juang, 2004), self-organizing hierarchical PSO with time-varying acceleration coefficients (HPSO-TVAC) (Ratnaweera, Halgamuge, & Watson, 2004), PSO with controllable random exploration velocity (PSO-CREV) (Chen & Li, 2007), and ACO (Cassillas et al., 2000). The number of rules in these methods is set to be four *a priori*, and the number of fuzzy sets in each input variable is equal to the fuzzy rule number. These comparing methods simultaneously design all centers and widths in each Gaussian fuzzy set and all consequent part parameters. Population size is also set at 40, and each method is performed for 50 runs.

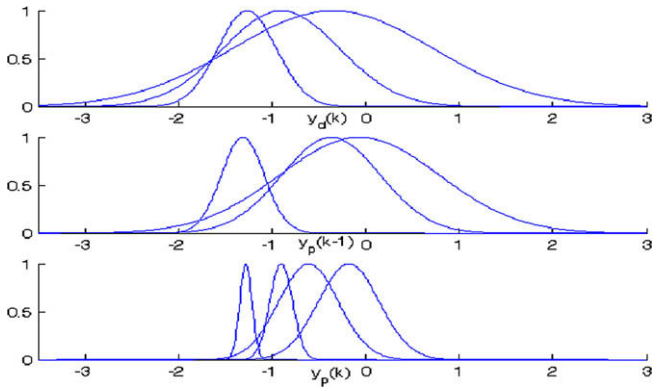

Fig. 5. The final distribution of fuzzy sets in each input variables after APSCO in Example 1.

The details of EGA can be found in Juang (2002).The crossover and mutation operations are the same as those used in Juang (2002). The mutation probability is set at 0.1 as that used in Juang (2002).

The compared CPSO uses Eq. (17) for particle velocity update. The particle update parameters are the same as those used in HGAPSO Juang (2004) and APSCO.

The details of HGAPSO can be found in Juang (2004). HGAPSO uses constriction PSO. The parameters for GA and CPSO update are the same as those used in Juang (2004).

The details of HPSO-TVAC can be found in Ratnaweera et al. (2004). The HPSO-TVAC uses typical PSO in Eq. (7) with time-varying acceleration coefficients (PSO-TVAC) and linearly decreasing inertia weight. According to Ratnaweera et al. (2004), the following simulations change c_1 from 2.5 to 0.5 and change c_2 from 0.5 to 2.5 over the whole search range. The inertia weight varies from 0.9 at the beginning of the search to 0.4 at the end of the search. The maximum velocity $v_{\max}$ is set to be 10 according to the search range and several trials.

The details of PSO-CREV can be found in Chen and Li (2007). The original parameters in Chen and Li (2007) works poorly in this example and a new set of parameters is selected. Parameters $c_1 = c_2 = 1$, $\alpha = 0.95$, $a = 1.5$, $b = 0.2$, $\eta = 0.993$ are selected. The range of $\bar{\xi}$ is selected to be $[-0.5, 0.5]$. The above parameters achieve the best performance from several trials.

For ACO, the antecedent part is partitioned in grid type as in Cassillas et al. (2000). Two Gaussian fuzzy sets exist in each input variable, which produces 8 rules in total. The shapes of these fuzzy sets are assigned in advance before learning. The scheme of using ACO for consequent part parameters design is the same as that used in Cassillas et al. (2000). The set of candidate control actions is the same as that used in APSCO.

Table 1 shows the final learning errors of the compared methods above, where it can be found that APSCO achieves the smallest control error. ACO produces the maximum control error because the search of the consequent parameters in discrete space degrades its tracking performance. The proposed APSCO relieves this constraint through PSO, and thus achieves greater accuracy.

5.2. Example 2 (Truck reversing control)

Fig. 7 shows the simulated truck. The truck position is precisely determined by the three state variables ϕ, x and y, where ϕ is the angle of the truck with the horizon as shown in Fig. 7. The angle of the control signal to the truck is θ, where $\theta \in [-20^\circ, 20^\circ]$. Only reversing is considered. The truck moves backward by a fixed unit distance every stage. The model of the truck reversing system is

$$x(k+1) = x(k) + \cos[\phi(k) + \theta(k)] + \sin[\theta(k)]\sin[\phi(k)] \tag{23}$$

$$y(k+1) = y(k) + \sin[\phi(k) + \theta(k)] - \sin[\theta(k)]\cos[\phi(k)] \tag{24}$$

$$\phi(k+1) = \phi(k) - \sin^{-1}\left[\frac{2\sin(\theta(k))}{b}\right] \tag{25}$$

Table 1
Comparison of the sum of absolute errors (SAE) of different methods in Example 1

Methods	ACO	EGA	CPSO	HGAPSO	PSO-CREV	HPSO-TVAC	APSCO[a]	APSCO[b]
Average	31.15	5.57	6.27	5.33	4.0	6.64	3.73	4.46
Std.	4.83	1.34	2.69	2.89	1..20	5.64	1.65	6.23

[a] Without self-aligning operation in the OSAC.
[b] With self-aligning operation in the OSAC.

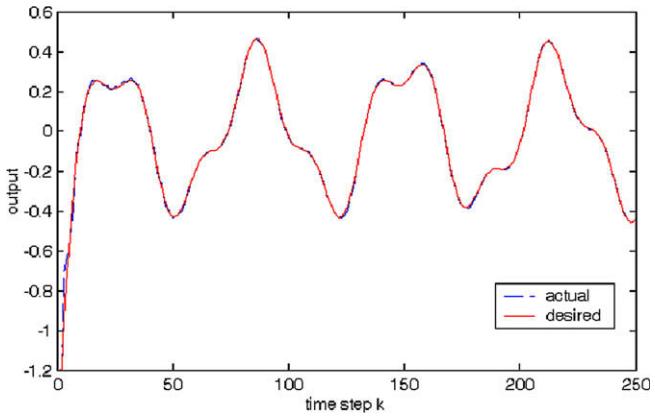

Fig. 6. The control results using APSCO in Example 1, where the desired and actual outputs are denoted by solid and dash lines, respectively.

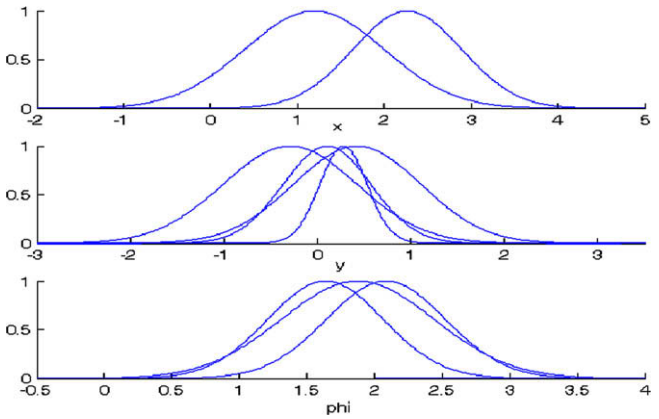

Fig. 8. The final distributions of the fuzzy set in input variables using APSCO in Example 2.

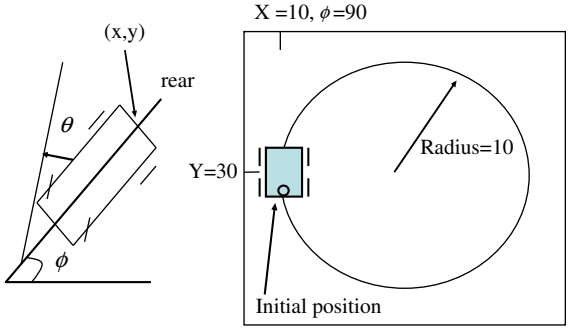

Fig. 7. Diagram of the simulation truck and the desired backing path.

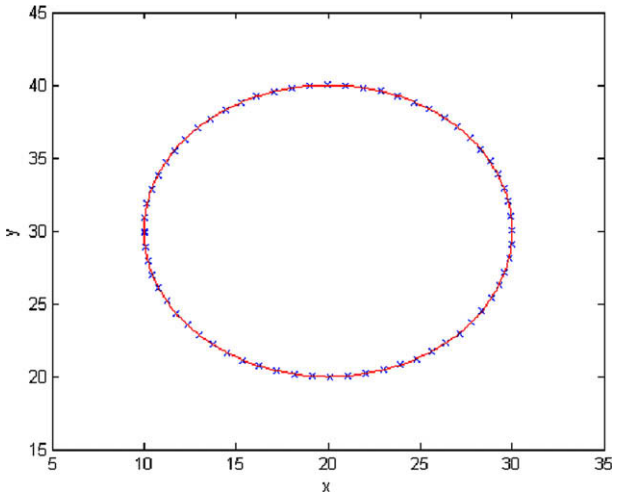

Fig. 9. The control results using APSCO in Example 2, where the desired and actual paths are denoted by solid line and "×", respectively.

where b is the length of the truck and $b = 4$ in this simulation. Eqs. (23)–(25) generate the next state when the present state and control are given. The task here is to design a fuzzy controller to reverse the truck following the desired circular path:

$$(x-20)^2 + (y-30)^2 = 10^2 \tag{26}$$

with an initial state of $(x, y, \phi) = (10, 30, 90^\circ)$. Fig. 7 shows the initial state and the desired driving path. There are a total of 65 control steps, and control error is defined as

$$\begin{aligned} \text{SAE} &= \sum_{k=1}^{65} |e(k)|, e(k) \\ &= (((x(k)-20)^2 + (y(k)-30)^2)^{\frac{1}{2}} - 10)^2 \end{aligned} \tag{27}$$

Fuzzy controller inputs are $x(k)$, $y(k)$, and $\phi(k)$. In OSAC, parameter ϕ_{th} for fuzzy clustering is set at 0.5. In APSCO, the set of candidate control actions is $U = \{0, \pm 1^\circ, \pm 2^\circ, \ldots, \pm 20^\circ\}$, with 41 candidate actions in the set. Another simulation with a smaller candidate set with 21 candidate actions in the consequent is also simulated. The pheromone update quality value is $F = 100/\text{SAE}$. The number of iterations is 1000, with 50 runs. The average number of fuzzy rules generated over these 50 runs is 4. The number of fuzzy sets with $s_{\text{th}} = 0$ and $s_{\text{th}} = 0.8$ are 12 and 9, respectively. An average of 2, 4, and 3 fuzzy sets occur in inputs $x(k)$, $y(k)$, and $\phi(k)$, respectively. Fig. 8 illustrates the final distribution of fuzzy sets in the three inputs after learning from one of the 50 runs. Table 2 shows the learning average and STD of the SAE. Fig. 9 shows the best control result. Table 2 shows the results, which indicate that there is only a little degradation in the average SAE when the consequent candidate actions decrease from 41 to 21.

The EGA, CPSO, HGAPSO, HPSO-TVAC, and PSO-CREV approaches described in Example 1 are also compared to APSCO. The numbers of rules in these compared methods are all set at 4, as in APSCO. Table 2 shows the learning results of these compared methods. As in the above example, APSCO achieves the smallest control error with a smaller population size than the compared methods.

Table 2
Comparison of the sum of absolute errors (SAE) of different methods in Example 2

Methods	EGA	CPSO	HGAPSO	HPSO-TVAC	PSO-CREV	APSCO (21)[a]	APSCO (41)[a]	APSCO (41)[b]
Average	0.525	0.5438	0.141	0.3252	1.29	0.109	0.088	0.112
Std.	0.564	1.0643	0.176	0.257	0.67	0.152	0.103	0.134

[a] Without self-aligning operation in the OSAC.
[b] With self-aligning operation in the OSAC.

6. Conclusion

This paper proposes combining OSAC and APSCO for the structure and parameter design of a fuzzy system. The technique of structure learning by clustering is often used in fuzzy neural networks instead of evolutionary or swarm intelligence fuzzy systems. This paper proposes the idea of incorporating OSAC into swarm intelligence fuzzy system design. Using OSAC not only helps determine the number of fuzzy sets, but also helps locate the initial antecedent parameters for subsequent parameter learning using APSCO. For parameter learning, the proposed APSCO method combines the search abilities of ACO and PSO by introducing auxiliary particles. The cooperative search of ACO and PSO compensates for the searching disadvantage of each optimization method. Simulation results show that the proposed method achieves a smaller deign error than the compared learning methods in each example.

References

Belarbi, K., & Titel, F. (2000). Genetic algorithm for the design of a class of fuzzy controllers: An alternative approach. *IEEE Transactions on Fuzzy Systems, 8*(4), 398–405.

Cassillas, J., Cordon, O., Viana, I.F. & Herrera, F. (2000). Learning fuzzy rules using ant colony optimization algorithms. In *Prof ANTS'2000 – from ant colonies to artificial ants: 2nd international workshop on ant algorithms* (pp. 12–21). September.

Chen, X., & Li, Y. (2007). A modified PSO structure resulting in high exploration ability with convergence guaranteed. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 37*(5), 1271–1289.

Chou, C. H. (2006). Genetic algorithm-based optimal fuzzy controller design in the linguistic space. *IEEE Transactions on Fuzzy Systems, 14*(3), 372–385.

Chung, I. F., Lin, C. J., & Lin, C. T. (2000). A GA-based fuzzy adaptive learning control network. *Fuzzy Sets and Systems, 112*, 65–84.

Clerc, M., & Kennedy, J. (2002). The particle swarm – explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation, 6*(1), 58–73.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 26*(1), 29–41.

Dorigo, M., & Stützle, T. (2004). Ant colony optimization. *MIT*. July.

Giordano, V., Naso, D., & Turchiano, B. (2006). Combining genetic algorithms and Lyapunov-based adaptation for online design of fuzzy controllers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 36*(5), 1118–1127.

Heo, J. S., Lee, K. Y., & Ramirez, R. G. (2006). Multiobjective control of power plants using particle swarm optimization techniques. *IEEE Transactions on Energy Conversion, 21*(2), 552–561.

Homaifar, A., & McCormick, E. (1995). Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. *IEEE Transactions on Fuzzy Systems, 3*(2), 129–139.

Höppner, F., Klawonn, F., Kruse, R., & Runkler, T. (1999). *Fuzzy cluster analysis: Methods for classification. Data analysis and image recognition*. John Wiley & Sons, Ltd.

Ho, S. L., Yang, S., Ni, G., & Wong, H. C. (2006). A particle swarm optimization method with enhanced global search ability for design optimizations of electromagnetic devices. *IEEE Transactions on Magnetics, 42*(3), 1107–1110.

Juang, C. F. (2002). A TSK-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithms. *IEEE Transactions on Fuzzy Systems, 10*(2), 155–170.

Juang, C. F. (2004). A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 34*(2), 997–1006.

Juang, C. F., & Lin, C. T. (1998). An on-line self-constructing neural fuzzy inference network and its applications. *IEEE Transactions on Fuzzy Systems, 6*, 12–32.

Juang, C. F., Lin, J. Y., & Lin, C. T. (2000). Genetic reinforcement learning through symbiotic evolution for fuzzy controller design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 30*(2), 290–302.

Kennedy, J. & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks* (pp. 1942–1948). Perth, Australia, December.

Kennedy, J., Eberhart, R. C., & Shi, Y. (2001). *Swarm intelligence*, San Francisco, Morgan Kaufmann.

Lin, C. T., & Lee, C. S. G. (1996). *Neural fuzzy systems: A neural-fuzzy synergism to intelligent systems*. Prentice Hall.

Mendes, R., Cortez, P., Rocha, M., & Neves, J. (2002). Particle swarms for feedforward neural network training. In *Proceedings of international joint conference on neural networks* (pp. 1895–1899).

Michalewicz, Z. (1999). *Genetic algorithms + data structures = evolution programs*. Springer.

Parpinelli, R. S., Lopes, H. S., & Freitas, A. A. (2002). Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computing, 6*(4), 321–332.

Ratnaweera, A., Halgamuge, S. K., & Watson, H. C. (2004). Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation, 8*(3), 240–255.

Shi, Y., & Eberhart, R. (1998). A modified particle swarm optimizer. In *Proceedings of IEEE world conference on computational intelligence* (pp. 69–73).

Sim, K. M., & Sun, W. H. (2003). Ant colony optimization for routing and load-balancing: survey and new directions. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 33*(5), 560–572.

Stützle, T., & Hoos, H. H. (2000). MAX MIN ant system. *Journal of Future Generation Computer Systems, 8*(16), 889–914.

Wong, C. C., & Her, S. M. (1999). A self-generating method for fuzzy system design. *Fuzzy Sets and Systems, 103*, 13–25.

Yao, X. (Ed.). (1999). *Evolutionary computation – theory and applications*. World Scientific.